НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису»
УДК 004.056

«До захисту допущено»
В.о. завідувача кафедри
_____ М.В.Грайворонський
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: Метод оптимізації мутаційного фазингу

Виконав (-ла): студент (-ка) 2 курсу, групи ФБ-71мп
(шифр групи)

Почвірний Микола Олегович
(прізвище, ім’я, по батькові)

Науковий керівник к.т.н., доц. Барановський Олексій Миколайович _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент к.т.н., ст.. викл. ОНАЗ Височіненко М.С. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (спеціалізація) – 125 Кібербезпека («Системи і технології кібербезпеки)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Почвірному Миколі Олеговичу

1. Тема дисертації: Метод оптимізації мутаційного фазингу

науковий керівник дисертації к.т.н., доц. Барановський Олексій Миколайович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15» листопада 2018 р. № 4171-с

2. Термін подання студентом дисертації 12.12.2018 р.

3. Об'єкт дослідження _____
_____

4. Вихідні дані _____
_____
_____
_____

5. Перелік завдань, які потрібно розробити _____
_____
_____
_____
_____
_____

6. Орієнтовний перелік ілюстративного матеріалу _____
_____
_____

7. Орієнтовний перелік публікацій _____
_____

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент _____ (підпис) _____ (ініціали, прізвище)

Науковий керівник дисертації _____ (підпис) _____ (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Робота обсягом 131 сторінку містить 21 ілюстрацію, 44 таблиці та 25 літературних посилань.

Метою даної магістрської роботи є підвищення ефективності методу мутаційного фазингу з отриманням практичних результатів його роботи.

Об’єктом дослідження є метод мутаційного фазингу.

Предметом дослідження є покращений метод генерації насіння мутаційного фазингу.

Методами дослідження було обрано: опрацювання літератури за даною темою, аналіз вразливостей та недоліків в існуючих методах, покрокова побудова метода генерації насіння фазингу.

Результати роботи можуть застосовуватись компаніями, що виготовляють програмне забезпечення для покращення роботи свого продуктц, зменшення помилок при його використанні та зменшення ціни розробки програмного забезпечення на етапі розробки та тестування.

ФАЗИНГ, МУТАЦІЙНИЙ ФАЗИНГ, ВРАЗЛИВОСТІ, НАСІННЯ ФАЗИНГУ, НЕЙРОННІ МЕРЕЖІ, МЕТОД ГЕНЕРАЦІЇ НАСІННЯ ФАЗИНГУ.

РЕФЕРАТ

Работа объемом 131 страницу содержит 21 иллюстраций, 44 таблицы и 25 литературных ссылок.

Целью данной квалификационной работы является улучшение эффективности метода мутационного фаззинга с получением практических результатов его работы.

Объектами исследования являются методы мутационного фаззинга.

Предметами исследования являются улучшенные методы генерации семян мутационного фаззинга.

Методами исследования были выбраны: проработка литературы по данной теме, анализ уязвимостей и недостатков в существующих методах, пошаговое построение метода генерации семян фаззинга.

Результаты работы могут применяться компаниями, которые производят программное обеспечение для улучшения работы своего продукта, уменьшения ошибок при его использовании и уменьшения цены разработки программного обеспечения на этапе разработки и тестирования.

ФАЗЗИНГ, МУТАЦИОННЫЙ ФАЗЗИНГ, УЯЗВИМОСТИ, СЕМЕНА ФАЗЗИНГА, НЕЙРОННЫЕ СЕТИ, МЕТОД ГЕНЕРАЦИИ СЕМЯН ФАЗЗИНГА.

ABSTRACT

The work includes 131 pages, 21 figures, 44 tables and 25 literary references.

The aim of this qualification work is effective improvement of mutational fuzzing with practical results of its work.

The object of research is mutational fuzzing method.

The subject of research is improved method of mutational fuzzing seed generation.

Methods of research: the analysis of the literature, the analysis of vulnerabilities and deficiencies in existing methods, step-by-step construction of fuzzing seed generation.

The results of the work can be used by companies that produce software to improve the performance of their products, reduce errors in the product and reduce cost of software development during development and testing steps.

FUZZING, MUTATIONAL FUZZING, VULNERABILITY, FUZZING SEEDS, NEURAL NETWORK, METHOD OF FUZZING SEED GENERATION.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів 9
Вступ 12
1 Аналіз існуючих методів фазингу 14
1.1 Метод Box-based фазингу 16
1.2 Фазинг з використанням знань про структуру вхідних даних 24
1.3 Метод фазингу з використанням різних типів генерування вхідних даних 28
1.4 Порівняння існуючих методів фазингу 40
Висновок до розділу 1 45
2 Порівняння існуючих методів ефективної генерації насіння фазингу 46
2.1 Метод генерації насіння фазингу - Smart Seed Generation for Efficient Fuzzing 47
2.2 Метод генерації насіння фазингу - Sheduling of Mutational Fuzing Seed 49
2.3 Метод генерації насіння фазингу - Program-Adaptive Mutational Seed Generation for Fuzzing 50
2.4 Метод генерації насіння фазингу - Data-Driven Seed Generation for Fuzzing 51
2.5 Порівняння методів генерації насіння фазингу 54
Висновок до розділу 2 57
3 Метод оптимізації мутаційного фазингу 58
3.1 Загальна архітектура системи генерації насіння для мутаційного фазингу 59
3.2 Особливості роботи системи генерації насіння 67
3.3 Особливості протокольного мутаційного фазингу 69
3.4 Особливості мережевого мутаційного фазингу 72
Висновок до розділу 3 73
4 Результати роботи методу генерації вхідних насінних файлів для мутаційного фазингу 74
4.1 Результати методу генерації вхідних насінних файлів для мутаційного фазингу для файлів 83

4.2 Результати методу генерації вхідних насінних файлів для мутаційного фазингу мережевих сервісів 89
4.3 Результати методу генерації вхідних насінних файлів для мутаційного фазингу мережевих протоколів 95
Висновок до розділу 4 100
5 Стартап 101
5.1 Опис ідеї проекту (товару, послуги, технології) 101
5.2 Технологічний аудит ідеї проекту 103
5.3 Аналіз ринкових можливостей запуску стартап проекту 103
5.4 Розроблення ринкової стратегії проекту 112
5.5 Розроблення маркетингової програми 114
Висновок до розділу 5 117
Висновки 117
Перелік джерел посилань 119
Додаток А 121
Додаток Б 127
Додаток В 132

Перелік умовних позначень, символів, одиниць, скорочень і термінів

Fuzzer – програма, що використовує методологію фазингового тестування.

Fuzzing – техніка автоматизованого тестування програмного забезпечення, яка полягає в тому, що на вхід програми подаються недійсні, невідповідні або випадково згенеровані дані.

Zero day arm race – характеризує протистояння розробників та хакерів в виявленні zero day вразливостей.

Zero day вразливості – вразливість програмного забезпечення, яка ще невідома користувачам чи розробникам програмного забезпечення та проти яких ще не розроблені механізми захисту.

Box based fuzzing – фазингове тестування з використанням структури скриньок.

SUT – System under test. Описує систему, що знаходиться під тестування на коректність.

Black-box – метод тестування функціональної поведінки об’єкта, без інформації про внутрішню побудову системи.

White-box – метод структурного тестування системи, з використанням її внутрішніх механізмів.

Grey-box – комбінація методів white-box та black-box, призначений для пошуку дефектів, які пов’язані з неправильною структурою, або неправильним використанням об’ктів тестування.

IG – input gain. Визначається як здатність вхідних данних знаходити нові вразивості, шляхом виконання нових базових блоків або збільшення частоти раніше виконуваних базових блоків.

GUI – graphic user interface. Графічний інтерфейс, створений для спрощення використання системи.

AST – abstract syntax tree. Скінченна множина, позначене і орієнтоване дерево, в якому внутрішні вершини співставлені з відповідними операторами мови програмування, а листя з відповідними операндами.

PCSG – probabilistic contextual-sensitive grammar. Контекстно-вільна граматика, в якій кожному правилу виведення відповідає ймовірність. Імовірність висновку визначається як добуток ймовірностей використовуваних в ньому правил виведення, таким чином, деякі висновки краще відповідають стохастичній граматиці, ніж інші.

COTS – commercial off the-shelf. Використовується для опису покупки упакованих рішень, які потім адаптуються для задоволення потреб організації, що купує, а не введення в експлуатацію замовлених або самостворених рішень.

FCS – fuzz configuration scheduling. Характеризує процес створення конфігурації для розпаралелювання етапів фазингу, з пошуком оптимальної втрати часу до знайдених вразливостей.

MAB – multi-armed bandit. Проблема, в якій фіксований обмежений набір ресурсів повинен бути розподілений між конкуруючими методами таким чином, щоб максимізувати їх очікуваний приріст, коли властивості кожного методу лише частково відомі під час розподілу, і можуть бути краще вивчені з часом або шляхом виділення ресурсів на вибір.

MITM – man in the middle. Модель атаки або тестування, при якій відбувається втручання в обмін інформацією між двома об’єктами, що думають, що вони контактують безпосердньо один з одним, з ціллю компрометації переданої інформації.

CRC – cyclic redundancy check. Алгоритм обчислення контрольної суми, призначений для перевірки цілісності даних.

DCGAN - Deep convolutional generative adversarial networks. Основа для навчання моделі DL для збору даних розподілу навчальних даних, щоб можливо було генерувати нові дані з цього самого розподілу.

WGAN – Wasserstein generative adversarial networks. WGAN це специфічний тип GAN, який, мінімізує розумне та ефективне наближення EM відстані, де EM відстань - це метод, який дозволяє подивитися на несхожість між двома багатовимірними наборами даних.

GAN - Generative adversarial networks. Генеративні змагальні мережі, що мають функції, пов'язані з дискримінацією між наборами даних та вибором результатів, є принципово корисними для машинного навчання.

CVE - Common Vulnerabilities and Exposures. CVE - це список записів, кожен із яких містить ідентифікаційний номер, а також опис та принаймні одну загальнодоступну довідку - для загальнодоступних вразливостей у сфері кібербезпеки.

PFOR – Predefined frame-of-reference. Ефективний метод копресії даних звикристанням FOR та дельта кодування.

DDSG – Data-Driven Seed Generation. Метод генерації насіння фазингу на основі авто згенерованих вхідних даних.

XSL - Extensible Stylesheet Language.XSL - це мова для вираження таблиць стилів.

ВСТУП

Назва магістрської роботи:

Метод оптимізації мутаційного фазингу

Обґрунтування та актуальність роботи:

У сучасних додатках актуальна тема вразливостей різного роду: переповнення буфера, витоку пам'яті, погане шифрування, недостатня перевірка вхідних даних. Деякі розробники програмного забезпечення в своїй діяльності успішно застосовують статичний аналіз коду (на етапі компіляції), а також динамічний (в процесі виконання коду). При аналізі компілюємого коду з точки зору безпеки, під динамічним аналізом часто мають на увазі саме фазинг. Перевагою фазинга є практично повна відсутність помилкових спрацьовувань, що досить часто зустрічається при використанні статичних аналізаторів. Фазинг - це технологія автоматизованого тестування програмного забезпечення з метою виявлення потенційних вразливостей, яка охоплює велику кількість граничних випадків шляхом породження некоректних вхідних даних. В якості вхідних даних при цьому можуть виступати оброблювані додатком файли, інформація, що передається по мережевим протоколам, функції прикладного інтерфейсу і т. д.

Однією із основних проблем напрямку фазингу є вузьконаправленість, тобто під кожну систему для ефективного етапу фазингу слід генерувати окремі насіння вхідних даних, та налаштовувати програму для фазингу під особливосі інфраструктури системи. Такі проблеми формують затримки в виконанні роботи системи та її відлагодженні, що є критичним фактором для подальшої оптимізації та поліпшенням ефективності роботи методу фазинга.

Мета роботи:

Метою даної роботи є підвищення ефективності мутаційного фазингу.

Об’єкт дослідження:

Методи мутаційного фазингу.

Предмет дослідження:

Покращений метод генерації насіння мутаційного фазингу.

Методами дослідження було обрано: опрацювання літератури за даною темою, аналіз переваг та недоліків в існуючих методах, покрокова побудова методу генерації насіння мутаційного фазингу.

Наукова новизна:

В результаті проведеної роботи було, удосконалено один з методів генерації насіння мутаційного фазингу та запропоновано підхід до генерації вхідних та їх обробки в незалежності від підтипу фазингу, що може використовуватися.

Практичне значення:

Результати роботи можуть застосовуватись, для тестування систем на вразливості з метою виявлення недоліків на етапі тестування та їх поетапного усунення.

План роботи:

- Огляд літератури та пошук інформації в мережі Інтернет;
- Аналіз сучасних методів фазингу;
- Виявлення недоліків та переваг в методах фазингу;
- Розробка оптимізованого методу фазингу на основі знайдених недоліків в сучасних методах фазингу

Розділ 1 Аналіз існуючих методів фазингу

В даному розділі буде розглянуто існуючі методи фазингу. Детально описано та розглянуто на прикладах існуючих інструментів фазингу методів фазингу, а також виділено переваги та недоліки кожного методу, проведене порівняння та аналіз загальних недоліків найсучасніших технологій фазингу.

Фазинг - це зазвичай автоматизований процес введення випадкових даних у програму та аналіз результатів для пошуку потенційно небезпечних помилок [1].

У сфері кібербезпеки фазинг - це зазвичай автоматизований процес пошуку критичних помилок програмного забезпечення шляхом випадкового впорядкування різних перестановок даних у цільовій програмі, доки одна з цих перестановок не виявить вразливості. Це старий, але все більш і більш поширений процес як для хакерів, які шукають уразливості для експлуатації, так і для захисників, які намагаються їх усунути. І в епоху, коли кожен може використовувати потужні обчислювальні ресурси, щоб бомбардувати цільову програму жертви непотрібними даними у пошуках помилки, це стало важливим направленням в zero-day arms race [2].

Порівняно з традиційною зворотною інженерією, це свого роду проста наука для дослідження. Викидаючи багато даних в програму, швидко змінюючи їх і покладаючись на моніторинг програмного забезпечення, для виявлення потенційних критичних вразливостей, а не ретельно відслідковувати потік даних, щоб знайти помилку. Це спосіб дуже швидкого пошуку помилок на моменті розробки та тестування ПО [3].

Залежно від того, де здійснюється маніпуляції з даними, фазинг розділяється на безліч категорій:

- Фазинг з використанням різних типів генерування вхідних даних
- Box based фазинг

- Фазинг з використанням знань про структуру вхідних даних[1]

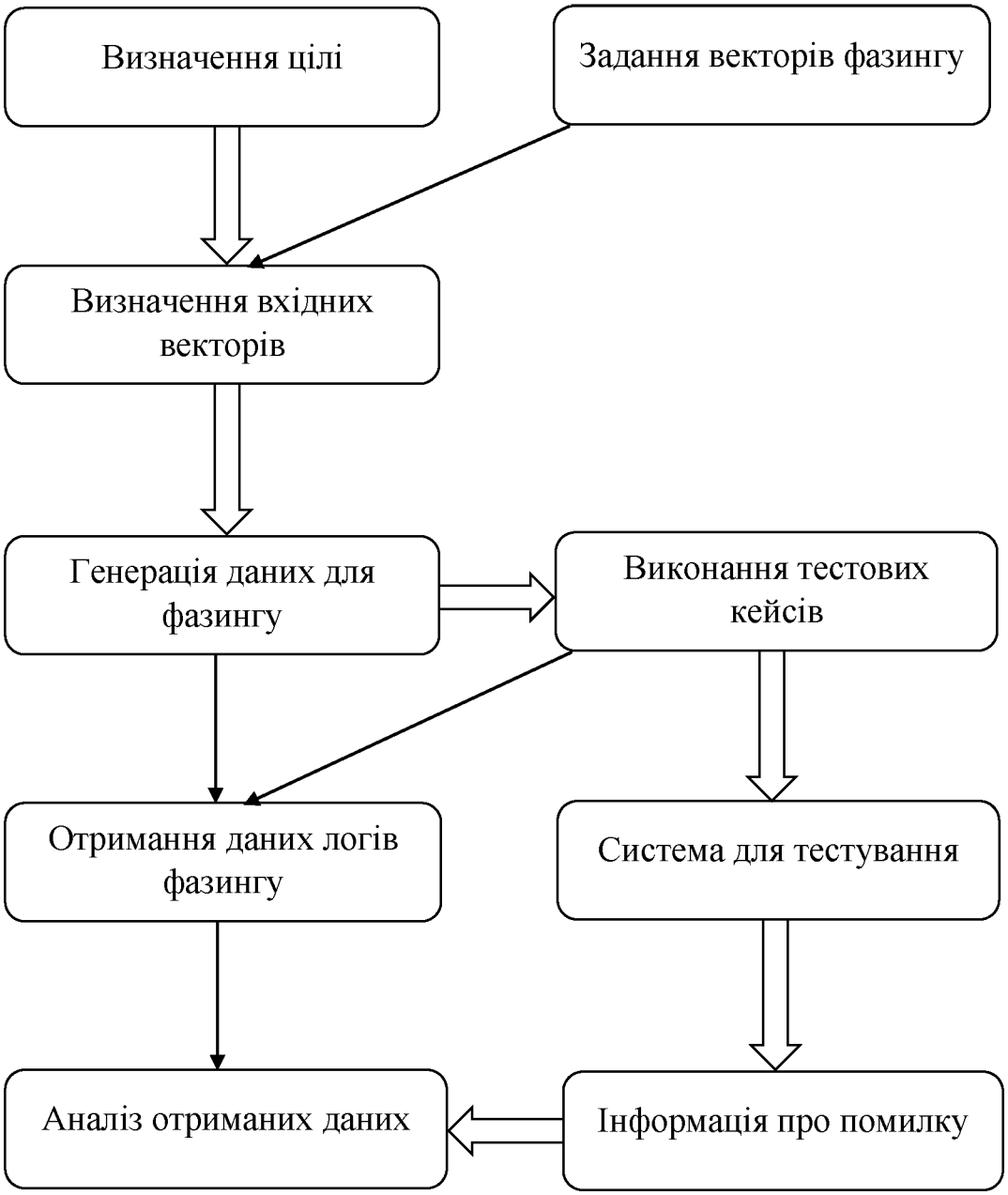

Рисунок 1.1 - Загальна схема фазингу

1.1 Метод Box-based фазингу

Основними категоріями програмного тестування є тестування чорного, сірого та білого ящиків. Різниця між цими категоріями визначається тим, який рівень інформації доступний для фазингу. На одному кінці є білий ящик, який вимагає повного доступу до всіх ресурсів, вихідного коду та специфікацій. На іншому кінці - чорний ящик, який не потребує знання внутрішніх технічних характеристик, і, отже, більше сліпий тест. У центрі - сірий ящик тестування. У сірому вікні може бути багато визначень залежно від ситуації, але це включає в себе тестувальника, який має доступ до деяких додаткових відомостей про SUT, наприклад, зібраних бінарних файлів. Фазинг падає головним чином у сіру зону коробки, тому що існує так багато різних підходів до дослыдження, що він може охоплювати весь спектр від білої до чорної скриньки [4,5].

Види box-based фазингу:

- Black-box
- White-box
- Grey-box

1.1.1 Метод White-box фазингу

White-box фазинг найчастіше використовується, як перевірка вихідного коду. Перевірка вихідного коду може бути виконана вручну, по рядку, але це непрактично для великих програм. Допомога автоматизованих інструментів потрібна для проходження всього коду, щоб знайти підозрілий код та потенційні помилки. Автоматизовані аналізатори вихідного коду можуть виявляти багато потенційних ризиків, що потрібно аналізувати, щоб вирішити, чи є вони реальною проблемою чи ні. Fuzzing може бути використаний як тест білої коробки для автоматичного тестування коду, щоб знайти справжні проблеми. Для використання цього методу найчастіше задається файл конфігурацій, де вказуються особливості вхідних даних, версії використаних framework, дані для мутації або статичні дані, глибина фазингу [4].

Недоліками є:

- Обов’язкова наявність повної інфрмації про систему
- Необхідність в заданні файлу конфігурацій
- Вузьконаправленість методу

Як приклад, представлено загальний принцип роботи white-box fuzzer SAGE. Він реалізує новий алгоритм прямого пошуку, який отримав назву generation search, що максимізує кількість нових вхідних тестів, створених за кожним символьним виконанням. Враховуючи обмеження шляху, всі обмеження на цьому шляху систематично відхиляються один за іншим, розміщуються в поєднанні з префіксом обмеження шляху, що призводить до нього, і намагається вирішити за допомогою обмежувача. Таким чином, єдине символічне виконання може генерувати тисячі нових тестів.

SAGE використовує декілька прийомів для покращення швидкості та використання пам'яті при генерації обмежень:

- кешування символічного виразу гарантує, що структурно еквівалентні символьні терміни відображаються на одному фізичному об'єкті;
- усунення несвязного обмеження зменшує розмір запитів на вирішення обмежень, видаливши обмеження, які не поділяють символічні змінні з негативним обмеженням;
- локальне кешування обмежень пропускає обмеження, якщо воно вже було додано до обмеження шляху;
- ліміт фліп - рахунку встановлює максимальну кількість разів, коли обмеження, створене з певної гілки програми, може бути перевернуто;
- використовуючи дешеву синтаксичну перевірку, обмеження підключення усуває обмеження, логічно підкріплені іншими обмеженнями, введені в одній і тій же програмі [5].

Рисунок 1.2 - Схема роботи White-box фазингу

1.1.2 Метод Black-box фазингу

Black-box фазинг проводиться, коли є програма, готова до використання, де можна вводити на вхід дані у вигляді команд, тексту, файлів, а вихід можна спостерігати. Іншими словами, black-box - це те, як кінцевий користувач використовує звичайне програмне забезпечення, немає ніякого знання про внутрішню роботу програми [1]. Фазинг Black-box - це динамічна методика пошуку помилок. Вона прагне знайти помилки в заданій програмі, виконуючи її за допомогою послідовності входів, що генеруються шляхом випадкового імітування вхідного вводу [2]. Програма, яка генерує ці входи та виконує програму на них, відома як fuzzer з black-box. В принципі, не існує жодних обмежень, крім того, що це строка з кінцевою довжиною;

однак на практиці часто вибирається як добре сформований набір вхідних даних для більш ефективного пошуку вразливостей. При кожному виконанні або збої спрацьовує, або закінчується належним чином. Однак декілька збоїв можуть бути пов'язані з тою самою основною помилкою. Таким чином, для виявлення кожного критичного виконання з відповідною помилкою, необхідно провести обробку помилок. Розуміння ефектів цих множинностей є ключовим моментом для аналізу black-box фазингу [4].

Недоліками є:

- Відсутність стабільного результату
- Складність покриття всіх вхідних даних
- Значний час відпрацювання

Як приклад, представлено загальний принцип роботи black-box fuzzer ansvif. При виконанні простого black-box фазингу, першочергово виконує тестування вхідних даних, для виявлення наявності фільтрування або строго лімітованого входу. На основі виявлених особливостей, починає генерувати нові програмні входи, з генерацією повного набору вхідних даних. В результаті відпрацювання буде перевірено всі можливі входи програми, що задовольняють обмеженням на вхід [6].

Рисунок 1.3 - Схема роботи Black-box фазингу

1.1.3 Метод Grey-box фазингу

Grey-box фазинг знаходиться прямо між white-box та black-box фазингами. Тестування сірої скриньки виконується, коли є більш глибоке знання системи, ніж простий чорний ящик, але не має доступу до вихідного коду або не має потреби робити white-box фазинг. Grey-box фазинг може включати в себе black-box із додатковим зворотним проектуванням двійкового файлу [7]. Grey-box фазинг може бути виконаний, якщо здійснюється деяка маніпуляція для двійкового файлу програми. Деякі інструменти для фазингу можуть вставляти інструменти в попередньо скомпільований двійковий файл, надаючи інформацію fuzzer. Найчастіше використовується при пошуку вразливостей в системних бібліотеках, драйверах, утилітах з вбудованими системами захисту [8].

Недоліками є:

- Потреба в модифікації двійкових файлів
- Відсутність стабільного результату
- Не ефективнй при тестуванні алгоритмів

Як приклад, представлено загальний принцип роботи grey-box fuzzer VUzzer. Він використовує еволюційну стратегію фазингу, еволюційний процес буде використовуватися для генерації входу до виникнення помилки або досягнення максимальної кількості поколінь. VUzzer зосереджується на створенні значущої інформації, яка вивчає нові цікаві шляхи в SUT. Термін "коефіцієнт надходження" (IG) використовується дослідниками VUzzer; IG вказує на здатність входу виконувати нові шляхи. VUzzer прагне генерувати ненульовий IG, відповівши на питання: де на вході мутувати і яке значення має бути там? За статичним та динамічним аналізом SUT VUzzer створює "розумний" зворотний зв'язок. Те, що дослідники вважають, що цикл є розумним, полягає в тому, що він розглядає функції керування та потоку даних з попереднього виконання та використовує цю інформацію для створення нового вводу. Функції потоку даних представляють інформацію про зв'язок між обчисленнями в межах SUT та вхідними даними. Витягуючи цю інформацію, VUzzer може вирішити питання про те, де і як мутувати. Функція керування потоком дозволяє обгрунтувати важливість шляху виконання. Зазвичай блокування обробки помилок не є важливими, і їх можна ігнорувати, тим самим пришвидшуючи генерацію вхідних даних, які будуть виконувати фузинги в більш релевантні частини SUT. За винятком ігнорування блоків обробки помилок, VUzzer може визначати пріоритети шляхів, які, швидше за все, заглиблюють виконання програми[9].

Рисунок 1.4 - Схема роботи Grey-box фазингу

1.2 Фазинг з використанням знань про структуру вхідних даних

Як правило, fuzzers використовуються для створення вхідних даних для програм, які приймають структуровані входи, такі як файл, послідовність дій клавіатури чи миші, або послідовність повідомлень. Ця структура відрізняє дійсний ввід, який приймається та обробляється програмою з недійсного вводу, який швидко відхиляється програмою. Те, що є дійсним вводом, може бути чітко визначено в моделі вводу. Прикладами вхідних моделей є формальні граматики, формати файлів, GUI-моделі та мережеві протоколи. Навіть елементи, які зазвичай не розглядаються як вхідні дані, можуть бути протестовані, наприклад, вміст баз даних, спільна пам'ять, змінні середовища або точне перемішування потоків. Ефективний fuzzer генерує напівпровідні вклади, які є "достатньо достовірними", так що вони не відхиляються безпосередньо від синтаксичного аналізатора та "недостатньо ефективні", щоб вони могли підкреслювати кутові справи та відстежувати незвичайну поведінку програми[1].

Фазинг з використанням знань про структуру вхідних даних поділяється на:

- Smart фазинг
- Dumb фазинг

1.2.1 Метод Smart фазингу

Розумний fuzzer використовує вхідну модель, щоб генерувати більшу частку дійсних входів. Наприклад, якщо вхід можна моделювати як абстрактне дерево синтаксису, то на базі інтелектуального мутаційного входу, fuzzer буде використовуватися випадкове перетворення для переміщення повних піддерев від одного вузла до іншого. Якщо вхідний сигнал може бути спроектований формальною граматикою, fuzzer на базі

інтелектуального покоління буде екземплятором правил виробництва для створення вхідних даних, які є дійсними щодо граматики. Проте в цілому вхідна модель повинна бути явно представлена, що важко зробити, коли модель є формальною, невідомою або дуже складною. Якщо наявний великий корпус дійсних та недійсних введень, то метод введення граматики, наприклад, алгоритм L * Англоуіна, зможе генерувати вхідну модель [10].

Недоліками є:

- Обов'язкове задання моделі вхідних даних
- Відсутність стабільного результату
- Потреба в розробці моделі під кожний окремий випадок

Як приклад, представлено загальний принцип роботи smart fuzzer Peach. Розширений і розширюваний, Peach fuzzer знаходить вразливості в апаратних та програмних рішеннях. Завдяки простому у використанні інтерфейсу на базі веб-інтерфейсу, великій бібліотеці підтримуваних протоколів та легко масштабованій платформі, Peach Fuzzer є провідним рішенням в галузі smart фазингу. Основними тестовими цілями платформи є:

- Користувачі файлів обробляють загальні формати файлів, будь то документ, зображення або потокове передавання
- Мережеві протоколи забезпечують зовнішні та внутрішні комунікації, документи, відеопотоки тощо
- Вбудовані пристрої включають в себе прошивку, емулятори та пристрої Android
- Драйвери пристроїв є важливими цілями для вибуху та є одними з найважчих цілей для тестування
- Зовнішні пристрої зростають у категорії "Інтернет", таких як автомобілі та інші бездротові пристрої[11]

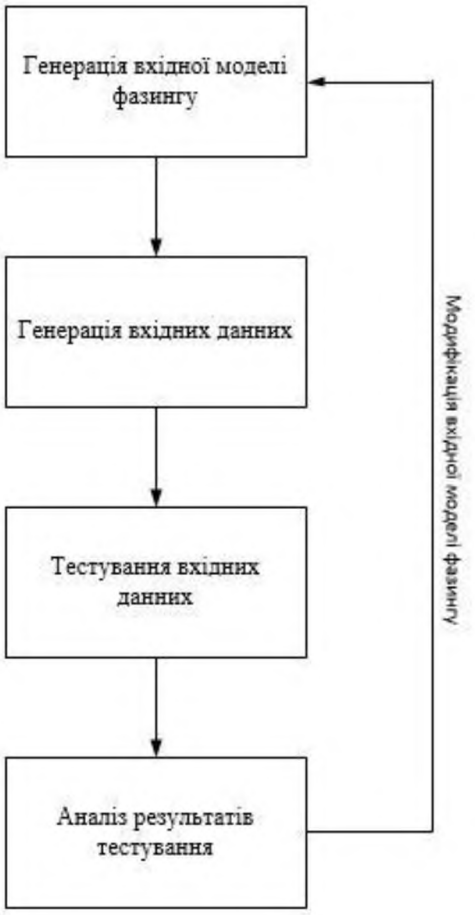

Рисунок 1.5 - Схема роботи Smart фазингу

1.2.2 Метод Dumb фазингу

Простий fuzzer не вимагає введення моделі, і тому може бути використаний для розповсюдження більш широкого спектру програм. Fuzzer, який генерує повністю випадковий ввід, відомий як "німий", так як він не має вбудованого знання про програму, вона проходить процес фазингу. Простий fuzzer вимагає найменшого обсягу роботи. Ця невелика кількість роботи може принести результати за дуже невелику вартість – одна з великих переваг фазингу. Однак іноді програма виконує лише певну обробку, якщо присутні певні аспекти вводу, як фільтрування вхідних даних або відсутність відповідних полів для обов’язкового заповнення [12].

Як приклад, представлено загальний принцип роботи smart fuzzer AFL. AFL - це fuzzer на основі німих мутацій, який змінює файл насіння шляхом перемикання випадкових бітів шляхом заміни випадкових байтів із особливими значеннями та переміщенням або видаленням блоків даних. Проте, німий fuzzer може генерувати меншу частку дійсних входів і відзначати недоліки коду, а не основні компоненти програми. Недоліком німого fuzzer можна проілюструвати за допомогою побудови дійсної контрольної суми для перевірки циклічної надмірності (CRC). CRC - це код для виявлення помилок, який забезпечує збереження цілісності даних, що містяться у вхідному файлі, під час передачі. Контрольна сума обчислюється над вхідними даними і записується у файл. Коли програма обробляє отриманий файл і фіксована контрольна сума не збігається з перевизначеною контрольністю, файл відхилено як недійсний. Тепер fuzzer, який не знає про CRC, навряд чи генерує правильну контрольну суму. Проте існують спроби ідентифікувати та перевизначити потенційну контрольну суму в мутованому введенні, коли fuzzer на базі німих мутацій змінив захищені дані [13].

Недоліками є:

- Обов’язкова генерація повної множини вхідних даних
- Довгий час відпрацювання
- Потреба великої кількості ресурсів

Рисунок 1.6 - Схема роботи Dumb фазингу

1.3 Метод фазингу з використанням різних типів генерування вхідних даних

При використанні fuzzer, на вхід завжди подаються дані, що можуть бути представлені в різному форматі та задані за допомогою різних методів. Серед них поширені статично сгенеровані вхідні дані та мутаційні дані, що поширюються від деякого набору вхідних даних. Статичні дані найчастіше використовуються при тестуванні програм з фільтруванням або складною структурою, при якій мутація не є ефективною, або при виявленні конкретних помилок в програмах, що характеризуються поданими на вхід даними. Мутаційні дані використовуються при наявності малої інформації про програму та її вхідних даних, що не дає можливості задати формат

вхідних даних для статичної генерації. Більшою популярністю, через універсальніть, користується метод мутаційного фазингу [1].

1.3.1 Метод Статичного фазингу

Ефективність статичного методу максимальна припускаючи, що програма приймає вхідні дані тільки в тому випадку, якщо вони є строго форматованими, даний алгоритм ніколи не генерує недоступні вхідні дані, тобто він уникає тупикових значень в лексеці та синтаксичному аналізі. Крім того, статичний метод може завершити частковий набір обмежень даних для повноцінного дійсного входу, таким чином, уникнути вивчення багатьох можливих недоповнюваних завершень. Обмежуючи пошуковий простір до дійсних введень, графічний фазинг з використанням білої скриньки може здійснювати більш глибокі шляхи і зосередити пошук на більш складних для випробувань, більш глибоких етапах обробки. Загалом при реалізації статичного фазингу порівняно з мутаційним, покриття коду виходить приблизно однакове, проте при цьому використовується менше тестів. Таким чином статичний фазинг має значні переваги в швидкості відпрацювання та використанні ресурсів для своєї роботи, при незначних втратах ефективності, порівняно з мутаційним фазингом [14].

Недоліками є:

- Великий об’єм даних для ефективної роботи
- Відсутність пошуку нових вразливостей
- Відсутність можливості тестування систем з динамічним відкликом

Як приклад, представлено загальний принцип роботи static input generation based fuzzer Skyfire. Він використовує величезну кількість зразків,щоб автоматично витягувати знання граматичних та семантичних правил, і використовує такі знання, щоб генерувати добре розподілені

витрати на насіння для програм, які обробляють високоструктуровані входи. У цьому сенсі Skyfire є ортогональним для футуристичних підходів на базі мутацій, забезпечуючи їм високоякісні насіння та підвищуючи їх ефективність та ефективність для програм, що обробляють високоструктуровані входи. Крім того, Skyfire розвиває існуючі підходи generation based фазингу, тобто несуть розвідку фазингу за межами семантичної перевірки, щоб досягти етапу виконання програми, щоб знайти глибокі помилки без будь-якого ручного втручання. Першочергово, Skyfire приймає в якості вхідних даних корпус і граматику, а також генерує вхідні дані з насіння в два етапи. Перший крок аналізує зібрані зразки на основі граматик в абстрактних дерева синтаксису (AST) і вивчає імовірнісну контекстно-чутливу граматику (PCSG), яка визначає як функції синтаксису, так і семантичні правила. На відміну від контекстно-вільної граматики, яка широко використовується в процесі фазингу на основі генерації, кожне правило виробництва в PCSG асоціюється з контекстом, в якому може застосовуватися правило виробництва, а також про ймовірність застосування правила виробництва в контексті. На другому кроці Skyfire спочатку генерує вхідні насіння, ітераційно вибираючи і застосовуючи правило виробництва, задовольняючи контекст, на нетермінальний символ, доки в результуючій рядку не буде нетермінального символу. Skyfire потім проводить виділення насіння, щоб відфільтрувати насіння, що мають те саме кодове покриття, щоб зменшити надмірність. Нарешті, Skyfire мутує залишкові насіння шляхом випадкового заміщення вузла листового рівня в AST з тим самим типом вузлів на основі правил виробництва, що різними способами впроваджують семантичні зміни при збереженні граматичних структур [15].

Рисунок 1.7 - Схема роботи статичного фазингу

1.3.2 Метод Мутаційного фазингу

Мутаційний фазинг - це один з найефективніших методів тестування при виявленні помилок безпеки та вразливостей у програмному забезпеченні Commercial Off The-Shelf (COTS). Це було величезним успіхом в практичних тестах безпеки, і його широко використовували великі програмні компанії, такі як Adobe та Google для цілей забезпечення якості. Ефективність фазингу багато в чому залежить від конфігурації насіння, яка являє собою набір параметрів для запуску fuzzer. Нещодавні дослідження, наприклад, показали, що кількість помилок, виявлених для однієї програми, при однакових обчислювальних ресурсах може сильно відрізнятися залежно від використовуваних файлів насіння. Основною проблемою є те, як знайти комбінацію параметрів fuzzing, що максимізує кількість виявлених помилок з урахуванням обмеженого ресурсу. Стан техніки для максимізації результату фізінгування полягає у пошуку за параметризованим простором фазингу, який називається Fuzz Configuration Scheduling (FCS). Тобто, fuzzers досліджують можливі комбінації параметрів і використовують часткову

інформацію, отриману в процесі розвідки, щоб максимально збільшити результат фазингу. Це класична компромісна "розвідка чи експлуатація", і це часто виражається як проблема Multi-Armed Bandit (MAB). На жаль, FCS є складним завданням, коли простір параметрів великий, оскільки існує забагато параметрів комбінації для розгляду. Наприклад, коефіцієнт мутації - швидкість між кількістю бітів для модифікації та кількістю загальних бітів насіння, яка використовується для обмеження відстані Хеммінга від насіння до сформованих тестових випадків, - це безперервний параметр, і таким чином він може мати безліч довільних значень [16,23].

Два методи, які можуть використовуватися мутаційними fuzzers:

- Replay
- MITM

1.3.2.1 Різновид методу Мутаційного фазингу - Replay

Фаззер може зберегти вхідні зразки та просто відтворювати їх після їх мутації. Це добре працює для фазингу формату файлів, де декілька зразків файлів можуть бути збережені і розфазовані для надання цільової програми. Прості або мережеві протоколи, що не зберігають стан, також можуть бути ефективно застосовані під час фазингу, оскільки для fuzzer не потрібно буде робити багато складних запитів, щоб глибоко проникнути в протокол. Для більш складного протоколу повторення може бути складнішим, оскільки fuzzer може знадобитися динамічно реагувати на програму, щоб продовжувалася робота, щоб глибоко проникнути в протокол, або протокол може бути просто незастосовним для повторного використання.

Однією з особливостей метода є відсутність потреби в генерації конфігурацій для ефективного відпрацювання фазингу. Часто при відпрацюванні на простих вхідних форматах, показує високий коефіцієнт відношення швидкості до ефективності [17].

Недоліками є:

- вузьконаправленість метода
- робота з простими вхідними даними
- необхідність мінімальної інформації про вхідні дані
- відсутність стабільного знаходження вразливостей

Як приклад, представлено загальний принцип роботи mutation based replay fuzzer AutoFuzz. Двигун фазингу модифікує протокол зв'язку між сервером і клієнтом, застосовуючи функції фазингу. Поточний набір функцій фазингу містить як детерміновані, так і недетерміновані функції. Детерміновані функції вставляють попередньо запрограмовані дані в GMS, такі як великі рядки, максимальне / мінімальне ціле значення та інші. Недетерміновані функції випадково пропускають статичні чи змінні поля даних GMS, приймати випадкові переходи в FSA і вставляти випадкові дані в GMS. Загалом, AutoFuzz намагається визначити поля окремих протокольних повідомлень за допомогою алгоритмів біоінформатики. Для того, щоб визначати поля повідомлень, аналогічні зразки повідомлень вирівнюються за допомогою декількох алгоритмів вирівнювання по строках, а їх консенсусні послідовності аналізуються, щоб зрозуміти початок і кінець полів повідомлення [18].

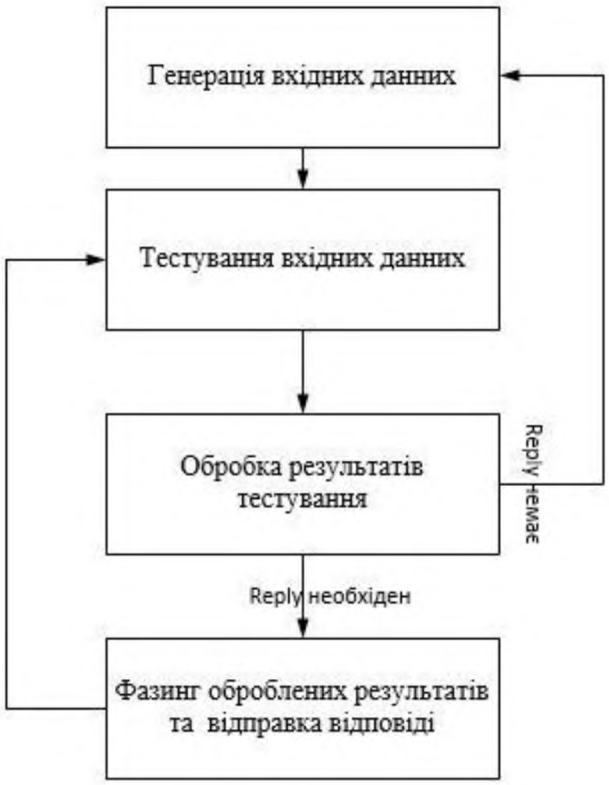

Рисунок 1.8 - Схема роботи Reply фазингу

1.3.2.2 Різновид методу Мутаційного фазингу - Man in the middle

MITM - це техніка, що використовується тестувальниками проникнення та хакерами, але вона також може використовуватися для фазування мережевого протоколу на базі мутацій. MITM описує ситуацію, коли fuzzer ставить себе посередині клієнта та сервера або двох клієнтів у випадку однорангової мережі, перехоплюючи та, можливо, модифікуючи передані між ними повідомлення. Таким чином, він діє як проксі між цими двома. Термін MITM зазвичай використовується, коли не очікується, що програма будете діяти як проксі-сервер, але для цілей фазингу терміни значною мірою взаємозамінні.

Встановивши fuzzer як проксі-сервер, він може змінювати запити чи відповіді, залежно від того, чи розривається з`єднання серверу та клієнту. Знову ж таки, fuzzer міг би не мати розвідки про протокол і просто випадково

змінювати деякі запити, або може цілеспрямовано змінювати запити на конкретному рівні протоколу, в якому буде проводитися фазування.

Фазинг на базі проксі-сервера дозволяє використовувати існуючу систему мережевої програми та швидко встановити в неї фазингове ядро, не вимагаючи, щоб fuzzer діяв як сам клієнт або сервер [17].

Недоліками є:

- Обов`язкове впровадження фазингового ядра
- Втручання в роботу між об`єктами системи
- Час відпрацювання залежить від налаштувань системи

Як приклад, представлено загальний принцип роботи mutation based man in the middle fuzzer AutoFuzz. Потік процесу, що включає в себе фазинг, використовуючи AutoFuzz, представлено в декілька етапів:

1. Протоколи трафіку записуються за допомогою вбудованого проксі-сервера AutoFuzz. Тілесони можуть вручну редагувати тестер, експортувати чи імпортувати в будь-який момент часу.
2. Модель поведінки протоколу побудована на основі пасивного синтезу з частковим скороченням автомата кінцевого автомата (FSA), запропонованим в ній.
3. Синтаксис окремих повідомлень витягується та зберігається в GMS. Продовжуючи використовувати функцію абстракції з GMS, щоб генерувати кластери вхідних повідомлень для побудови GMS. Отже, кожен кластер являє собою набір подібних вхідних повідомлень. З урахуванням функції абстракції для вхідних повідомлень, подібні вхідні повідомлення кластеризуються разом за допомогою affinity propagation функції абстракції. Потім алгоритми вирівнювання послідовностей застосовуються для створення GMS для кожного кластера. Далі, проходячи через протокол FSA, кожен перехід пов'язується з відповідним GMS.

4. Фазингові функції застосовуються шляхом модифікації сеансів живого спілкування між клієнтом і сервером. Фазинг двигун відповідає за присвоєння функції фазингу. Яка функція фазингу виконується, визначається поточним станом в FSA, вхідним повідомленням, які функції вже були застосовані [18].

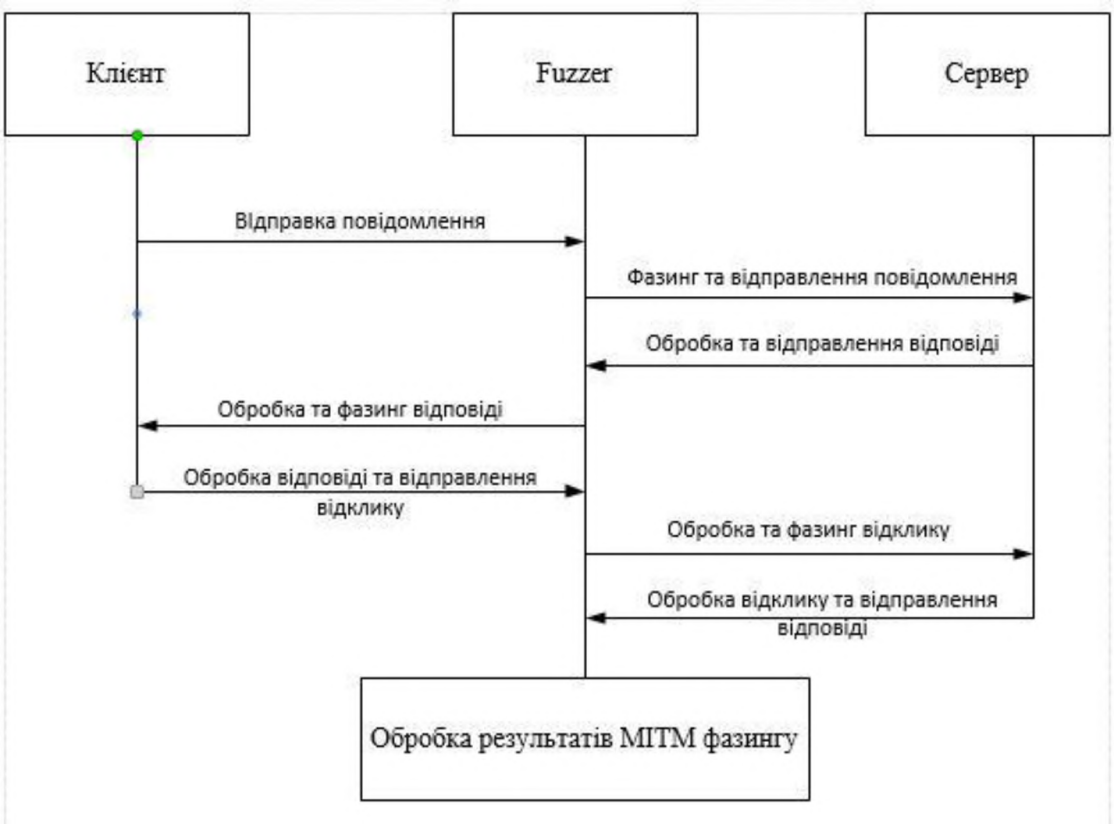

Рисунок 1.9 - Схема роботи MITM фазингу

1.3.3 Метод Generation-based фазингу

Fuzzers на основі генерації насправді генерують дані з нуля, а не мутують існуючий ввід. Fuzzers на основі генерації зазвичай вимагають певного рівня інтелекту для побудови вхідних даних, що принаймні має деякий сенс для програми, хоча генерація повністю випадкових даних також технічно буде генерацією.

Fuzzers на основі генерації часто розбивають протокол або файл формату на шматки, які вони можуть створити в правильному порядку, і випадковим чином виконують фазинг деякі з цих шматків самостійно. Це може створити входи, які зберігають загальну структуру, але містять суперечливі дані в цій структурі. Деталізація цих шматків та інтелект, з якою вони побудовані, визначають рівень інтелекту fuzzer [19].

У той час як мутаційний фазинг може мати подібний ефект, оскільки генерація є фазингом, оскільки з часом мутації будуть випадково застосовані без повного порушення структури вводу, генерація входів гарантує, що це буде так. Фазинг на основі генерації також може більш глибоко проникнути в протокол легше, оскільки він може будувати дійсні послідовності входів, застосовуючи фазинг до конкретних частин цього зв'язку. Це також дозволяє fuzzer виступати як справжній клієнт / сервер, створюючи правильні, динамічні відповіді, де їх не можна відслідковувати повторно [19].

Недоліками є:

- Необхідність знання про формат вхідних даних
- Необхідність грамотного розбиття елементу фазингу
- Необхідність генерації вхідних даних під кожний випадок

Як приклад, представлено загальний принцип роботи мультифункціонального generation-based fuzzer Peach. Peach fuzzer - це розумний fuzzer з можливістю генерації та мутації. Він працює шляхом створення файлів PeachPit, які є файлами XML, що містять повну інформацію про структуру даних, інформацію про тип та відношення даних.

Peach працює на декількох компонентах:

- моделювання даних;
- моделювання станів;
- видавець;

- агенти;
- монітори та логери та ін.[20]

Peach сильно зосереджується на моделюванні даних та моделюванні стану. Рівень деталізації, який вводиться в ці моделі, служить ілюстрації різкої різниці між dumb Peach fuzzer і smart.

Файли Peach Pit міститимуть принаймні одну модель даних. Моделі даних визначають структуру блоку даних, вказавши додаткові дочірні елементи, такі як Число та Строка.

Комплексні протоколи поділяються на частини: кожна частина з власною моделлю даних для повторного використання. Далі модель може бути розділена на розділи. Крім того, якщо додається посилання (ref-атрибут), вміст посилання буде скопійовано, щоб створити базу нової моделі даних. Будь-які дочірні елементи в моделі даних замінять елементи, які вже існують з таким самим ім'ям [21].

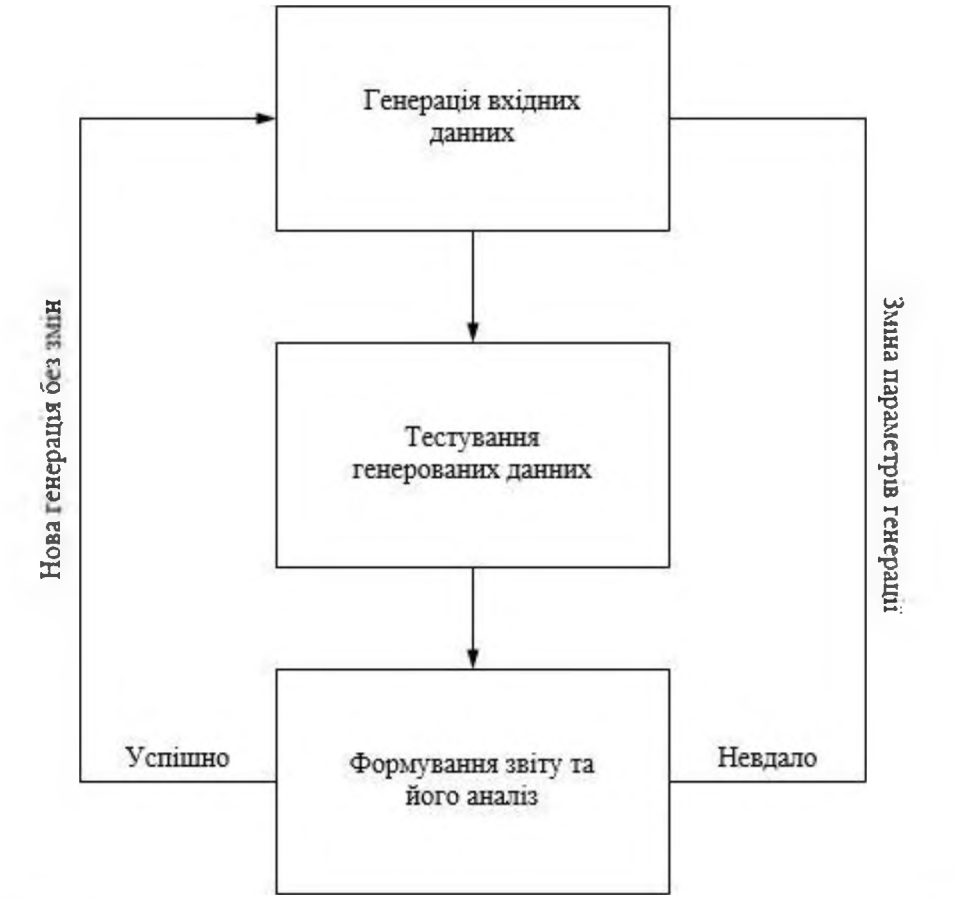

Рисунок 1.10 - Схема роботи Generation-based фазингу

1.4 Порівняння існуючих методів фазингу

Black box фазинг використовується найчастіше при тестуванні систем, інформація про які недоступна, а white box фазинг навпаки, для ефективності відпрацювання має знати повну інформацію про систему.

Grey box фазинг використовується в тих випадках, коли недостатньо інформації для описаних вище методів або система потребує в проведенні змін в двійкових виконуваних файлах для тестування, наприклад системні драйвери або бібліотеки.

Smart фазинг складно віднести до будь-якого з наведених вище методів, проте найчастіше його відносять до методів похідних від white box, особливістю є обов'язкова генерація моделі тестування та відповідного набору конфігурацій для вхідних даних, тоді як dumb фазинг відноситься більше до black box фазингу та не потребує генерації моделей та конфігурацій, часто пов'язується з мутаційним фазингом проте мутаційний фазинг знаходиться між smart та dumb фазингом.

Мутаційний фазинг можна охарактеризувати, як комбінацію простого smart фазингу та black box фазингу, адже під час проведення тестування відбувається мутація вхідних даних на основі роботи попередніх вхідних даних, при цьому немає жодної інформації про систему, а конфігурація вхідних даних задається не користувачем, а програмою в процесі тестування.

Статичний фазинг являє собою похідний метод від white box фазингу, основною особливістю є те ще в процесі тестування не відбувається змін вхідних даних, а ціллю даного методу є виявленя конректних широко поширених або вузьконаправлених вразливостей, а не пошук нових, що є відрізняючою особливістю даного методу від інших.

Generation-based фазинг один з методів, що часто використовують для генерації даних в smart фазингу, оскільки для ефективності роботи smart

фазингу слід подавати на вхід строго форматовні дані, а найефективнішим способом є їх генерація на основі попередніх поколінь, порівняно іншими методами є не стільки методологією фазингу, як підходом до генерації даних фазингу.

Таблиця 1.1 – Порівняльний аналіз методів фазингу

Назва методу	Переваги	Недоліки
Black box	Мінімальна кількість інформації для роботи. Ефективність при пошуку нових вразливостей. Можливість тестувати системи в незалежності від вхідних даних.	Відсутність стабільного результату. Складність покриття всіх вхідних даних. Значний час відпрацювання. Не ефективний в системах з строго форматованими вхідними даними.
White box	Ефективний в системах з повністю доступною інформацією. Швидкий час відпрацювання. Завжди видає стабільний результат.	Обов'язкова наявність повної інформації про систему. Необхідність в заданні файлу конфігурацій. Вузьконаправленість методу. Не можливий в системах з мінімальною інформацією.

Продовження таблиці 1.1

Назва методу	Переваги	Недоліки
Grey box	Ефективний при тестуванні системних драйверів та бібліотек. Можливість пошуку нових вразливостей. Можливість знаходження вразливостей в програмному коді.	Потреба в модифікації двійкових файлів. Відсутність стабільного результату. Не ефективнй при тестуванні алгоритмів.
Smart	Ефективний при тестуванні систем з відкритою інформацією. Чим більше інформації, тим вище ефективність. Швидкість відпрацювання після генерації моделі. Ефективний при тестуванні систем з строго форматованим вхідними даними.	Обов’язкове задання моделі вхідних даних. Відсутність стабільного результату. Потреба в розробці моделі під кожний окремий випадок. Не ефективний при тестуванні систем з мінімальною інформацією.

Продовження таблиці 1.1

Назва методу	Переваги	Недоліки
Dumb	Ефективний при тестуванні будь-яких систем. Пошук всіх можливих вразливостей. Простий в використанні. Не потребує автоматизаії.	Обов'язкова генерація повної множини вхідних даних. Довгий час відпрацювання. Потреба великої кількості ресурсів.
Статичний	Ефективний при тестуванні алгоритмів. Можливість пошуку будь-яких відомих вразливостей. Ефективність прямо пропорційна кількості наявних даних.	Великий об'єм даних для ефективної роботи. Відсутність пошуку нових вразливостей. Відсутність можливості тестування систем з динамічним відкликом.
Replay	Можливість тестування систем з динамічним викликом. Ефективний при тестуванні мережевих протоколів. Швидкість відпрацювання залежить тільки від таймаутів в середині системи.	Вузьконаправленість метода. Робота з простими вхідними даними. Необхідність мінімальної інформації про вхідні дані. Відсутність стабільного знаходження вразливостей.

Продовження таблиці 1.1

Назва методу	Переваги	Недоліки
MITM	Можливість тестування систем з динамічним відкликом. Ефективний при тестуванні мережевих протоколів та систем. Не потребує великих об’ємів вхідних даних.	Обов'язкове впровадження фазингового ядра. Втручання в роботу між об`єктами системи. Час відпрацювання залежить від налаштувань системи.
Generation-based	Швидкість виконання та генерації даних. Ефективний при тестуванні систем з строго форматованими вхідними даними. Можливість побудови моделі вхідних для якісної генерації.	Необхідність знання про формат вхідних даних. Необхідність грамотного розбиття елементу фазингу. Необхідність генерації вхідних даних під кожний випадок.

Висновок до розділу 1

В даному розділі були розглянуті методи фазингу.

Можна зробити висновок, що на сьогоднішній день проблема виявлення вразливостей в програмах є однією з найважливіших в сфері інформаційних технологій. Щодня на ринок інформаційних технологій виходить все більше і більше продуктів для широго користування. В результаті чого для ефективного пошуку вразливостей, постійно розробляються і вдосконалюються різні методи. Одним із таких методів є фазинг, який з плином часу набув популярності та виділяється своєю ефективністю серед інших методів. Однак на основі проведеного аналізу методів фазингу, можна однозначно охарактеризувати їх, як не досконалі, з своїми недоліками та перевагами, а отже існує можливість їх оптимізації та поліпшення, або створення нових методів на основі комбінації існуючих.

Розділ 2 Порівняння існуючих методів ефективної генерації насіння фазингу

В даному розділі буде представлено порівняння різних методів генерації насіння фазингу. Також буде описано базові характеристики методів, що на даний момент є актуальними рішеннями в сфері мутаційного фазингу загалом та як методи генерації насіння мутаційного фазингу.Серед найбільш поширених слід виділити так методи:

- **Smart Seed Generation for Efficient Fuzzing**
- **Sheduling of Mutational Fuzing Seed**
- **Program-Adaptive Mutational Seed Generation for Fuzzing**
- **Data-Driven Seed Generation for Fuzzing**

2.1 Метод генерації насіння фазингу - Smart Seed Generation for Efficient Fuzzing

В основному робочий потік SmartSeed складається з трьох етапів.

1. Підготовка. SmartSeed - це система машинного навчання. Щоб завантажити SmartSeed, треба підготувати необхідні навчальні дані. Зокрема, збираються деякі регулярні файли та використовуються, щоб визначати деякі програми за допомогою звичайно використовуваних фазингових інструментів, таких як American Fuzzy Lop (AFL). Потім збираються файли введення, які викликають унікальні збої або нові шляхи, як дані тренування. Цей крок використовується лише для збору необхідних даних для підготовки до завантажувального SmartSeed (який потім може бути використаний для генерації насіння для виправлення багатьох застосувань), і може бути легко застосований на практиці [22].
2. Модельне будівництво. Щоб зробити SmartSeed легко розширюваним на практиці, пропонується механізм перетворення для кодування необоротних навчальних даних у загальні матриці, які потім використовуються для побудови генеративної моделі для генерації насіння. Використовуючи генеративну модель, генеруються ефективні файли як насіння [22].
3. Фазинг. Використовуючи насіння, створене з побудованої генеративної моделі, використовуются фазингові інструменти (наприклад, AFL) для виявлення збоїв об'єктиних застосувань. Весь процес може сформувати замкнений цикл. Модель машинного навчання може генерувати ефективні фільтри насіння, щоб допомогти фазинговим інструментам виявити нові збої та шляхи об'єктивних застосувань. Потім, навчальний комплект моделі машинного навчання в SmartSeed може бути відновлений і покращений додатковими файлами введення, які викликають нові збої або шляхи [22].

2.2 Метод генерації насіння фазингу - Sheduling of Mutational Fuzing Seed

Мутаційний фазинг Black-box - це динамічна технологія виявлення помилок. Він прагне виявляти помилки в заданій програмі p, використавши його на послідовність входів, сформованих шляхом випадкового мутування певного вхідного сигналу s. Програма, яка генерує ці входи та виконує p на них, відома як мутаційний fuzzer з black-box. Загалом, не існує жодних обмежень на s, крім того, що це ланцюжок з кінцевою довжиною; однак, на практиці часто вибирається як добре сформований вхід для p в інтересах виявлення помилок більш ефективно. З кожним виконанням p будь-який збій або належним чином закінчується. Однак декілька збоїв можуть бути пов'язані з тим самим основним помилкою. Таким чином, для виявлення кожної аварії у відповідну помилку необхідно провести обробку помилок. Розуміння ефектів цих множинності є ключовим моментом для аналізу мутаційного спалаху чорної коробки. Щоб формально визначити мутаційний фазинг в black-box, потрібне поняття "випадкові мутації" для бітових рядків. У подальшому нехай | s | позначає бітову довжину s [23].

Основною ідеєю при генерації насіння є виконання випадкової мутації з початковими входами та кореляцією в залежності від відпрацювання алгоритму:

Мутаційний fuzzer з black-box - це рандомізований алгоритм, який приймає в якості введення конфігурацію даних, яка включає - програму p, вхідний сигнал s і коефіцієнт мутації $r \in [0,1]$. У режимі run fuzzer генерує вхід x шляхом випадкового мутації s з мутаційним співвідношенням r і потім запускає p на x. Результатом цього фазингу є крах або правильне припинення роботи p.

В даному алгоритмі під випадковою мутацією слід розуміти:

Випадкова мутація біта b є комбінація біта b і рівномірно обраним бітом з логікою виключного або. Що стосується даного співвідношення мутації r ∈ [0,1], то випадкова мутація строки s генерується шляхом першого вибору d = r · | s |. Різновид бітових позицій рівномірно випадковим серед | s | d можливих комбінацій, а потім випадково змінювати ці d-біти в s. Мутації у формі безумовного встановлення або відключення біта можливі, однак їх обидва важче аналізувати математично і рідше застосовується на практиці, через не контрольну генерацію даних та можливість повного порушення структури об'єкта[23].

2.3 Метод генерації насіння фазингу - Program-Adaptive Mutational Seed Generation for Fuzzing

В даній моделі кожен вхід має фіксовану довжину N бітів. Вхідний простір I_N позначає всесвіт усіх можливих входів розмірів N біт. Тому потужність вхідного простору $| I_N | - 2^N$. Є вхідні дані у вхідному просторі, які ініціюють одне або кілька програмних помилок, які називаються помилковими виходами. Для вказування бітової позиції на вході використовується індекс. Наприклад, s_1 означає перший біт входу s. Відстань Хеммінга - кількість різниць бітів у двох вхідних рядках - між входом і та j у вхідному просторі за δ (i, j) [24].

Визначення 1 (K-сусіди). K-сусід входу і довжини N являє собою вхід, відстань Хеммінга від і дорівнює K. Позначення множини всіх K-сусідів і за N_{Ki}: $N_{Ki} = \{j \in \{0, 1\}^N \mid \delta (i, j) = K\}$.

З огляду на наведене вище визначення, можна стверджувати, що два набори сусідів одного входу з іншим значенням K незмінні один від одного: $\forall i,\ 0 \leq A, B \leq N: N_{Ai} \cap N_{Bi} = \emptyset \Leftarrow \Rightarrow A \neq B$. μ - функція, яка приймає як тестовий тест і набір бітових позицій як вхідний матеріал, і повертає

видозмінений тестовий випадок, коли кожний певний біт перетворюється (комбінація ексклюзивного або з 1) з даним тестовим випадком [24].

Наприклад, μ (s, {3,4}) - це вхід, в якому обидва третій та четвертий біти s flipped і δ (μ (s, {3,4}), s) = 2. Представляючи виконання як послідовність інструкцій. З урахуванням програми p і введення s seed для програми, виконання p за допомогою s як введення σ_p (s). Функція оцінки береться за виконання програми і виводить або помилку , або успішне виконання. Припускається, що ідентифікатор помилки однозначно визначає вид аварій.

Наприклад, якщо програма p збирається, коли виконується тест i, то:

∂ (δ_p (i)) = c, де c - це ідентифікатор помилки [24].

2.4 Метод генерації насіння фазингу - Data-Driven Seed Generation for Fuzzing

З урахуванням накопиченого DDSG, можна генерувати набір насіннєвого вводу через лівостороннє виведення.Зокрема, для генерації вхідного t ми можна спочатку встановити t на початок символу DDSG і ітеративно застосувати наступні кроки до тих пір, поки не буде нетермінального символу в t:

1. отримати найменший нетермінальний символ l у t і відповідний контекст c;
2. випадковим чином вибирається правило виробництва r з R_l, ліворуч - l, заданий c;
3. застосувати r на l в t.

Метод цього покоління, який базується на випадковому лівому виведенні, припиняється, коли досягнуто бюджетний час, або створюється задана кількість насіннєвого матеріалу. Через характер DDSG більшість

отриманих даних правильні щодо граматик і семантичних правил. t відкидається, якщо у нього все ще є не термінали, коли витрачається часовий бюджет [15].

З огляду на DDSG $G_p = ((N, \Sigma, R, s), q)$, лівий висновок являє собою послідовність символів $t_0, ..., t_n$, де $t_0 = s$, тобто t_0 містить тільки початковий символ.

- $t_n \in \Sigma^*$, тобто tn складається з термінальних символів, а Σ^* позначає набір всіх можливих рядків, складених послідовностями слів, взятих з Σ.
- t_i для $i = 1, ..., n$ виводиться з t_{i-1} шляхом заміни лівого нетермінального символу α (з контекстом c) в t_{i-1} з певним β, де $[c]\alpha \rightarrow \beta$ є виробничим правилом у R.

Проте, через випадковість, що змінює виробничі правила в методі вищезгаданої генерації, можуть виникнути деякі проблеми.

По-перше, покоління може стати нескінченним, оскільки правила виробництва можуть бути рекурсивними. Отже, розширення нетермінального символу за допомогою застосування рекурсивного правила виробництва може призвести до рядка, що включає той самий нетермінальний символ знову. Наприклад, елемент у XSL може містити вміст, а вміст може містити ще один елемент. Далі ця проблема зростає, коли правило виробництва може містити багато нетермінальних символів. Наприклад, вміст може містити сотні елементів [15].

По-друге, створювані вклади можуть стати непотрібно складними через рекурсивні правила виробництва (тобто збільшення складності в глибині) або багато нетермінальних символів (тобто збільшення складності в ширину). В результаті отримані дані можуть бути дуже великими за розмірами, які можуть бути відхилені безпосередньо fuzzers(наприклад, AFL відмовляється брати вхідні дані, які за замовчуванням перевищують 1 Мб).

Щоб вирішити ці проблеми та слідувати головним цілям, пропонується метод генерації, який базується на евристичному лівому виведенні.

Алгоритм 1 Генерування вхідних елементів насіння з входу DDSG:

Input: the PCSG $G_p = ((N, \Sigma, R, s), q)$
Output: the set of seeds generated T
1: $T := \emptyset$
2: **repeat**
3: $t := s$ // set the seed input to the start symbol
4: $c := null$ // set the context to null
5: $num := 0$ // set the times of applying rules to 0
6: **repeat**
7: $l :=$ the left-most non-terminal symbol in t
8: update c according to l
9: $R_l :=$ the set of rules in R whose left-side is l given c
10: **if** $random() < 0.9$ **then**
11: heuristically choose a less-frequently applied and less-complexity rule r from low-probability rules in R_l
12: **else**
13: heuristically choose a less-frequently applied and less-complexity rule r from high-probability rules in R_l
14: **end if**
15: replace l in t with the right side of r
16: $num := num + 1$
17: **until** there is no non-terminal symbol in t, or $num == 200$
18: $T := T \cup \{t\}$
19: **until** time budget is reached, or enough seed inputs are generated

Алгоритм 1 описує процедуру методу. Метод попереднього випадкового генерації полягає в тому, що розробляються декілька евристичних елементів у лівому найбільш похідному (рядки 10, 11, 13 і 17) для евристичного вибору правила r з R_l, ліворуч - l, заданий контекст c (рядок 7-9) [15].

Евристика1: Фабрично-імовірність виробничих правил. Для того, щоб згенерувати непотрібне введення інформації про непередбачувану поведінку програми, використовується ймовірність від кожного правила до першого розділу R_1 до правил високої ймовірності R_H та правил низької ймовірності R_L , а потім вибирається правило відповідно до Евристик 2 і 3 від R_L з більшою ймовірністю (тобто 0,9 лінія), ніж з R_H l (лінія 10-13).

Евристика 2: Вибір низькочастотних правил генерації та обмеження кількості програм одного і того ж правила генерації. Найпопулярніші елементи, які використовуються як частково, так і генерують виразні входи, що мають різноманітні граматичні структури, записується інформація про історію застосування правила, тобто частоту, до якої було застосовано правило при створенні насіння.

Евристика 3: Вигідність правил генерації низької складності. Щоб зменшити непотрібну складність у великій кількості отриманих введених даних, використовується кількість нетермінальних символів у правилі, щоб виміряти складність правила та виступати за правила низької складності.

Евристика 4: Обмежити загальну кількість правил входів. Щоб зменшити непотрібну складність по глибині згенерованих введених даних, емпірично обмежується загальна кількість додатків правил при створенні входу (рядок 17) [15].

2.5 Порівняння методів генерації насіння фазингу

Переваги методу **Smart Seed Generation for Efficient Fuzzing:**

- Простота генерації вихідних даних
- Можливість генерації даних будь-якого формату
- Швидкість та ефективність навчання на вхідних даних
- Ефективність виявлення помилок для файлового фазингу
- Відсутність проблеми зациклювання при мутації даних

Недоліки методу **Smart Seed Generation for Efficient Fuzzing:**

- Відсутність гарантованого виявлення вразливостей
- Повторюваність знайдених помилок

Переваги методу **Sheduling of Mutational Fuzing Seed:**

- Знайдені помилки є унікальними
- Ефективність виявлення помилок на основі параметричних даних
- Ефективність виявлення помилок для протокольного фазингу.

Недоліки методу **Sheduling of Mutational Fuzing Seed:**

- Складність генерації вихідних даних
- Швидкість навчання на вхідних даних
- Швидкість генерації вихідних даних
- Необхідність строго-форматованого формату вхідних даних
- Присутність проблеми зациклювання при мутації даних.

Переваги методу **Program-Adaptive Mutational Seed Generation for Fuzzing:**

- Знайдені помилки є унікальними
- Ефективність виявлення помилок для файлового фазингу
- Ефективність виявлення помилок для мережевого фазингу

Недоліки методу **Program-Adaptive Mutational Seed Generation for Fuzzing:**

- Складність генерації вихідних даних
- Швидкість генерації вихідних даних
- Необхідність строго-форматованого формату вхідних даних
- Можливість частого циклічного повторення даних згенерованих на виході.

Переваги методу **Data-Driven Seed Generation for Fuzzing:**

- Ефективність виявлення помилок для файлового фазингу
- Знайдені помилки є унікальними
- Відсутність проблеми зациклювання при мутації даних

Недоліки методу **Data-Driven Seed Generation for Fuzzing:**

- Складність генерації вихідних даних
- Швидкість генерації вихідних даних
- Необхідність строго-форматованого формату вхідних даних
- Повторюваність знайдених помилок
- Відсутність гарантованого виявлення вразливостей

На основі наведених вище переваг та недоліків можна визначити, що незважаючи на простоту метод Smart Seed Generation for Efficient Fuzzing, демонмтрує найбільшу ефективність при вирішенні поставленої проблеми. Виходячи з цього даний метод буде обраний як основа для методу описаного в розділі 3.

Висновок до розділу 2

В даному розділі були розглянуті порівняння різних методів генерації насіння фазингу.

В результаті був вибраний метод Smart Seed Generation for Efficient Fuzzing, як найефективніше рішення поставленої проблеми.

Зроблено висновок, що описаний в даному розділі метод частково вирішує поставлену проблему і є ідеальною основою для методу, що повністю покриє поставлену проблематику.

Розділ 3 Метод оптимізації мутаційного фазингу

В даному розділі буде описано метод оптимізації мутаційного фазингу за допомогою ефективнішого методу генерації насіння. Також буде детально описано та покроково роз'яснено особливості побудови метода в залежності від вибраного підтипу фазингу.

Для оптимізації існуючих методів фазингу є два основні підходи:

1. оптимізація алгоритмів фазингу;
2. оптимізація меоду генерації насіння фазингу;

Оптимальним рішенням є оптимізація генерації насіння, оскільки правильно сгенероване насіння покращує роботу будь-якого методу фазингу в декілька разів. На основі цього в даному розділі буде представлено загальну архітектуру генерації насіння для файлового, мережевого та протокольного фазингу.

3.1 Загальна архітектура системи генерації насіння для мутаційного фазингу

Основною ідеєю методу є побудова генеративної моделі. Тоді ми використовуємо цю модель для швидкого генерування цінних файлів як набору наборів вхідних засобів для розмивання. Як показано на рисунку 3.1, вся архітектура може бути розділена на 4 процедури:

(1) Збір навчальних даних: вводиться критерій для вимірювання значення вхідних файлів і представляється спосіб отримання набору для навчання.

(2) Перенесення необоротних даних: для вирішення проблем із нефіксованими форматами або нефіксованими розмірами файлів, перетворення бінарних файлів сировинних навчальних даних на єдиний тип матриць.

(3) Модель побудови: Визначення матриці в якості навчальних даних, побудова генеративної моделі насіння, заснованих на генеративних сусідніх мережах Wasserstein.

(4)Зворотне перетворення: на основі генеративної моделі, генерація нових матриць та перетворення їх у відповідні файли введення, що є зворотним процесом процедури (2).

У системі задіяний фазинговий інструмент може бути прозорим, тобто може поєднуватися з більшістю існуючих на базі мутацій фазингових інструментів.

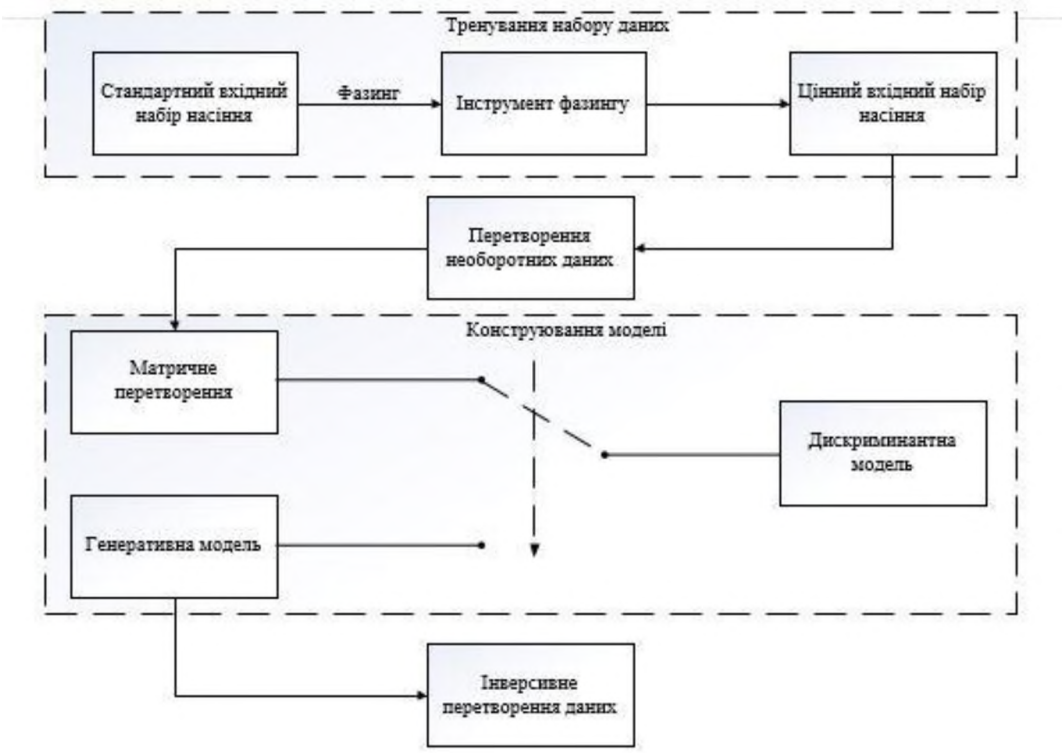

Рисунок 3.1 - Архітектура роботи моделі генерації насіння

3.1.1 Колекція навчальних даних для генерації насіння фазингу

Щоб побудувати модель машинного навчання для генерування цінних насіннєвих фільтрів, потрібно отримати набір первинної підготовки. Тому спочатку просіюються цінні вхідні файли. Зокрема, в даному випадку визначаються цінні файли як файли вводу, які викликають унікальні збої або унікальні шляхи застосування. Причини такі: (1) оскільки кінцевою метою фазингу є виявлення більшої кількості помилок, вхідні файли вважаються цінними, якщо вони можуть викликати унікальні збої об'єктивних застосувань; (2), згідно з існуючими дослідженнями збільшення охоплення та глибини шкідливих шляхів, швидше за все, збільшить кількість досліджених аварій. Отже, файли, що викликають нові шляхи, також є цінними з цієї точки зору. Можна використовувати існуючі стратегії вибору насіння для вибору кількох цінних файлів введення як набору тренувань. Проте, згідно існуючих досліджень та результатів експериментів, сьогоднішні стратегії

вибору насіння виглядають ненадійними. Потім ми усвідомлюємо, що такі інструменти. Таким чином, генерується така стратегія збору даних щодо тренувань: спочатку використовувати регулярні вхідні файли, зібрані з Інтернету, для розповсюдження програм з тим самим форматом введення. Потім збираються цінні вхідні файли, які ініціюють унікальні шляхи або унікальні збої цих програм, як навчальний набір.

3.1.2 Перетворення необоротних даних з колекції навчальних даних

Для побудови генераторної генерації моделі пропонується механізм перетворення вихідних файлів вхідних даних у навчальний набір до єдиного типу матриць. Причини проведення такої конверсії є наступними. По-перше, однією з цілей є створення методу узгодження з декількома форматами введення та необмеженими розмірами файлів. Однак незручно налаштовувати режим читання даних для різних типів і різних розмірів файлів. Таким чином, потрібно визначити єдиний спосіб читання даних з навчального набору. По-друге, формати багатьох файлів у наборі тренувань пошкоджені. Нормальний спосіб читання, наприклад, читання файлу bmp як тривимірної матриці, може не працювати в багатьох сценаріях застосування. По-третє, на підставі знань, що алгоритми машинного навчання краще працюють на кількісних значеннях матриць, а не на деяких випадкових величинах, потрібно виявити спосіб, який може перетворити декілька типів файлів у єдиний тип матриць. Оскільки таким чином для моделі машинного навчання буде легше вивчити особливості спеціальних байтів, які контролюють шляхи виконання коду. Таким чином, файли слід читати в двійковій формі і, як очікується, потрібне перетворення в єдину систему матриць. Вводяться основні процедури перетворення сирої обробки даних, які показані на рисунку 3.2 (1) Оскільки всі файли можуть бути прочитані в двійковій формі, можна прочитати бінарну форму будь-якого типу файлу і

отримати бінарний рядок файлу. (2) Для вирішення проблеми, як розпізнати кінець двійкового рядка, рядок кодується за допомогою Base64. Таким чином, є рядок, що складається з 64 видів символів і "=", наприклад, рядок символів, показаний на рисунку 3.2 (3) Тепер, оскільки рядок символів може мати 65 різних символів, для зручності відбувається перетворення символів від Base64 і "=" до чисел (від 0 до 64). Таким чином, отримується рядок чисел, як показано на рисунку 3.2. Механізм кодування для такого перетворення показаний на рисунку 3.3. (4) Щоб економити кількість елементів у матриці, робиться перетворення кожних шести цифр рядка в великі числа десяткової системи. Тоді ці числа десяткової системи нормалізуються на [0, 0.4902227890624] (з 65^7 -1 = 4902227890624) для точності та ефективності моделі навчання. Нарешті, додається 0 до кінця матриці, якщо є порожній елемент.

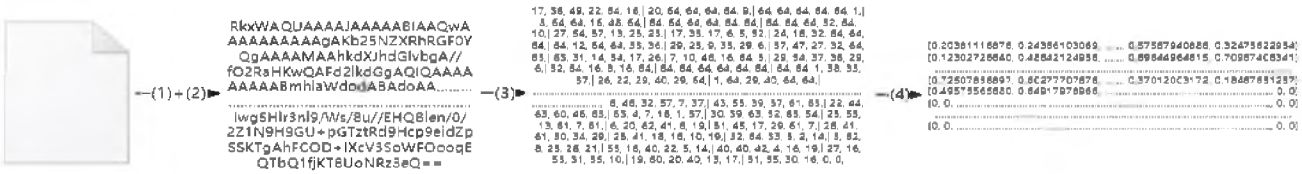

Рисунок 3.2 - Перетворення файлу вхідних даних до єдиної системи матриць

"A" = 0	"B" = 1	"C" = 2	"D" = 3	"E" = 4
"F" = 5	"G" = 6	"H" = 7	"I" = 8	"J" = 9
"K" = 10	"L" = 11	"M" = 12	"N" = 13	"O" = 14
"P" = 15	"Q" = 16	"R" = 17	"S" = 18	"T" = 19
"U" = 20	"V" = 21	"W" = 22	"X" = 23	"Y" = 24
"Z" = 25	"a" = 26	"b" = 27	"c" = 28	"d" = 29
"e" = 30	"f" = 31	"g" = 32	"h" = 33	"i" = 34
"j" = 35	"k" = 36	"l" = 37	"m" = 38	"n" = 39
"o" = 40	"p" = 41	"q" = 42	"r" = 43	"s" = 44
"t" = 45	"u" = 46	"v" = 47	"w" = 48	"x" = 49
"y" = 50	"z" = 51	"0" = 52	"1" = 53	"2" = 54
"3" = 55	"4" = 56	"5" = 57	"6" = 58	"7" = 59
"8" = 60	"9" = 61	"+" = 62	"/" = 63	"=" = 64

Рисунок 3.3 - Перетворення вхідних даних в форматі base64 до числового виду

3.1.3 Конструювання моделі для генерації насіння на основі перетворених необоротних даних

Одним з найкращих існуючих генеративних моделей є модель Generative Adversarial Networks (GAN), яка широко використовується для безконтрольного навчання з 2014 року. GAN - це нова система, яка складається з генеративної моделі та дискримінаційної моделі. Генеративна модель намагається створити підроблені дані, схожі на реальні дані в наборі тренувань, тоді як дискримінаційна модель намагається відрізнити підроблені дані від реальних даних. Дві моделі альтернативно працюють разом, щоб навчати один одного, і далі вдосконалювати один одного. В результаті, генеративна модель буде генерувати дані, які занадто реальні, щоб розрізняти дискримінаційні моделі. Зазвичай генеративна модель, надана GAN, може генерувати більш реалістичні дані, ніж інші алгоритми. GAN також має багато проблем, таких як колапс моделі. Оптимізованою моделлю GAN є Deep Convolutional GAN(DCGAN). DCGAN може генерувати більш реалістичні дані, ніж стандартні GAN, і це набагато простіше для тренувань.

Тим не менш, у нього все ще є багато проблем, включаючи колапс моделі. У 2017 році була представлена модель Wasserstein GAN (WGAN). На відміну від інших моделей GAN, WGAN покращує стабільність навчання. Модель WGAN тренувати простіше ніж її попередників. Крім того, WGAN може вирішити проблеми з режимом фазингу GAN у більшості сценаріїв застосування.

Отже, для мутаційного фазингу, буде використовуватися модель WGAN, щоб вивчати характеристики цінних файлів, а потім генерувати цінні насіннєві файли. Даний вибір також має наступні переваги:

- WGAN може вивчити особливості самого навчання. Таким чином, нам не потрібно звертати увагу на вибір функції, що є дуже ефективним.
- Ще однією перевагою є те, що можна вільно вибирати відповідну модель машинного навчання як генеративну модель та дискримінаційну модель WGAN.
- Аналізуючи Multi-Layer Perceptron (MLP), дана модель більше зосереджується на кожному кількісному значенні в матриці, в той час як Convolutional Neural Network(CNN) приділяє більше уваги глобальним особливостям матриці
- Щоб побудувати кращу модель, перевіряється продуктивність обох моделей нейронної мережі.

3.1.4 Інверсивна конвертація необоротних даних для отримання вихідного насіння фазингу

У цьому підрозділі описується, як використовувати генеративну модель для отримання ефективного набору насіння. Оскільки навчальний набір - це ряд матриць, генеративна модель навчається створювати подібні матриці. Щоб отримати двійкові вхідні файли для фазингу, потрібно перетворити сформовані матриці у двійкові файли. Таким чином, робиться зворотна сторона вищезазначених процедур у розділі 2.1.2. Щоб бути конкретним, першим кроком є відновлення елементів матриці великим числам десяткової системи. По-друге, перетворюється кожне число десяткової системи на шість цифр (від 0 до 64), тобто рядок цифр, як показано на рисунку 2.2. Потім перетворюються числа (від 0 до 64) на символи Base64 і "= ", Тобто рядок символів, як показано на рисунку 2.2. Нарешті, декодування символьного рядку Base64 в двійковий файл і зберігання його локально. У цьому випадку отримуються вхідні файли для мутаційного фазингу.

3.2 Особливості роботи системи генерації насіння

Роботу алгоритму можна поділити на декілька основних етапів:

1. Отримання базових вхідних данних для першого етапу фазингу. Вхідні файли представляють з себе данні, що не викликають вразливостей при фазингу відповідних систем і мають розмір до 64 кб, оскільки при перенасиченні нейронних мереж великою кілікістю даних, навчання буде неефективним. Ці файли будуть еталонними та будуть використовуватися для визначення, який формат даних потрібен для цілі фазингу, протокол, файл, текст ,або мережеві команди передані в протоколі.
2. Перша генерація файлів фазингу з базових вхідних файлів. Проведення фазингу на основі згенерованих даних і отримання файлів, що генерують помилки при роботі програми, протоколу або сервісу. З цих файлів вибираються файли, що більші за розміром від 10 кб і менші за 64 кь, оскільки при навчанні і подальшому отриманні еталонних файлів фазингу слід використовувати певну кількість інформації для генерації, не багато, щоб система могла визначити елементи фазингу, і не мало, щоб система могла отримати достатньо інформації для подальшої генерації.
3. Другий етап генерації файлів фазингу з файлів отриманих при фільтруванні файлів фазингу отриманих при першій генерації. Особливістю є те, що розмір файду на даному етапі генерації не обмежений ні чим, і для тестування системи використовуються найрізноманітніші вхідні дані.
4. При недостатньо ефективному відпрацюванні файлів згенерованих в другому етапі, виконується третій, четвертий... етап, при цьому вхідні дані для генерації кожний раз доповнюються файлами, що викликали

помилки при роботі програми, протоколу або сервісу, проте на даному етапі жодних обмежень на розмір вхідних даних використано не буде.

Описані вище особливості системи генерації насіння для мутаційного фазингу дозволять, якісним чином покращити виявлення помилок при тестуванні програмного забезпечення та систематизувати підхід до генерації насіння фазингу, з підвищенням ефективності генерації, не покладаючись на рандомізовані потенційні помилки.

3.3 Особливості протокольного мутаційного фазингу

Особливості будуть продемонстровані на прикладі протоколу UDP. На вхід системи першочергово буде поданий протокол у форматі поданому на рисунку 3.4

```
Eth:  ******* Ethernet - "Ethernet" - offset=? length=70
Eth:
Eth: destination = 44:1c:a8:81:07:17
Eth:      source = 50:c7:bf:e5:d4:44
Eth:        type = IPv4 (0x0800)
Eth:
IP:  ******* IPv4 - "Internet Protocol (Version 4)" - offset=? length=56
IP:
IP:                version = IPv4
IP:           header length = 5 bytes
IP: differentiated services = 0x10
IP:                           0001 00.. = [4] code point
IP:                           .... ..0. = [0] ECN
IP:                           .... ...0 = [0] ECE
IP:             total length = 56
IP:           identification = 0x0 (0)
IP:                    flags = 0x02
IP:                           0.. = [0] reserved
IP:                           .1. = [1] don't fragment
IP:                           ..0 = [0] more fragments
IP:          fragment offset = 0
IP:             time to live = 48
IP:                 protocol = UDP (0x11)
IP:          header checksum = 0x56cd [valid]
IP:                   source = 74.125.232.71
IP:              destination = 192.168.0.107
IP:
UDP:  ******* UDP - "User Datagram Protocol" - offset=? length=36
UDP:
UDP:      source = 443
UDP: destination = 64517
UDP:      length = 36
UDP:    checksum = 0xefc9 [valid]
UDP:
```

Рисунок 3.4 - Вхідний формат для протоколу UDP

Наступний етап – визначення параметрів, що будуть змінюватися в протоколі при генерації насіння фазингу.

Дані, що будуть змінюватися в процесі фазингу рандомним чином, будуть показані системі між квадратними дужками [].

Перший метод – фазинг базових найменувань в протоколі, перевірка наявості валідації протоколу. Представлено на рисунку 3.5.

```
Eth:  ******* Ethernet - "Ethernet" - [offset]=? [length]=70
Eth:
Eth: [destination] = 44:1c:a8:81:07:17
Eth:      [source] = 50:c7:bf:e5:d4:44
Eth:        [type] = IPv4 (0x0800)
Eth:
IP:  ******* IPv4 - "Internet Protocol (Version 4)" - [offset]=? [length]=56
IP:
IP:                [version] = IPv4
IP:           [header length] = 5 bytes
IP: [differentiated services] = 0x10
IP:                           0001 00.. = [4] code point
IP:                           .... ..0. = [0] ECN
IP:                           .... ...0 = [0] ECE
IP:             [total length] = 56
IP:          [identification] = 0x0 (0)
IP:                   [flags] = 0x02
IP:                           0.. = [0] reserved
IP:                           .1. = [1] don't fragment
IP:                           ..0 = [0] more fragments
IP:         [fragment offset] = 0
IP:            [time to live] = 48
IP:                [protocol] = UDP (0x11)
IP:         [header checksum] = 0x56cd [valid]
IP:                  [source] = 74.125.232.71
IP:             [destination] = 192.168.0.107
IP:
UDP:  ******* UDP - "User Datagram Protocol" - [offset]=? [length]=36
UDP:
UDP:      [source] = 443
UDP: [destination] = 64517
UDP:      [length] = 36
UDP:    [checksum] = 0xefc9 [valid]
UDP:
```

Рисунок 3.5 - Виділення базових елементів протоколу

Після виділення основних елементів протоколу, дані конвертуються до формату base64, перетворюються в числовий формат, переводяться до формату великих чисел і подаються на вхід моделі для генерації виходу.

Другий метод – фазинг параметрів протоколу для виявлення потенційних помилок. Представлено на рисунку 3.6

```
Eth:  ******* Ethernet - "Ethernet" - offset=[?] length=[70]
Eth:
Eth: destination = [44:1c:a8:81:07:17]
Eth:      source = [50:c7:bf:e5:d4:44]
Eth:        type = [IPv4 (0x0800)]
Eth:
IP:  ******* IPv4 - "Internet Protocol (Version 4)" - offset=[?] length=[56]
IP:
IP:                 version = [IPv4]
IP:           header length = [5 bytes]
IP: differentiated services = [0x10]
IP:                           [0001 00.. = [4] code point
IP:                           .... ..0. = [0] ECN
IP:                           .... ...0 = [0] ECE]
IP:             total length = [56]
IP:           identification = [0x0 (0)]
IP:                    flags = [0x02]
IP:                           [0.. = [0] reserved
IP:                           .1. = [1] don't fragment
IP:                           ..0 = [0] more fragments]
IP:          fragment offset = [0]
IP:             time to live = [48]
IP:                 protocol = [UDP (0x11)]
IP:          header checksum = [0x56cd [valid]]
IP:                   source = [74.125.232.71]
IP:              destination = [192.168.0.107]
IP:
UDP:  ******* UDP - "User Datagram Protocol" - offset=[?] length=[36]
UDP:
UDP:      source = [443]
UDP: destination = [64517]
UDP:      length = [36]
UDP:    checksum = [0xefc9 [valid]]
UDP:
```

Рисунок 3.6 - Виділення параметрів базових елементів протоколу

Після виділення параметрів базових елементів протоколу, дані конвертуються до формату base64, перетворюються в числовий формат, переводяться до формату великих чисел і подаються на вхід моделі для генерації виходу. При строгій валідації протоколу, параметр checksum розраховується окремо та змінюється після генерації.

3.4 Особливості мережевого мутаційного фазингу

Можна поділити на декілька основних етапів:

1. Генерація базових мережевих протоколів, що викликають помилку при роботі сервісу.
2. Генерація бази даних команд, що будуть передаватися в протоколі по мережі.
3. Генерація бази даних параметрів стандартних вразливостей для кожної команди.
4. Відпрацювання різних комбінацій команд та параметрів, що передаються різними протоколами, для визначення того, які протоколи підтримує система, які команди можливі для виконання в системі, чи вразлива система на базові вразливості.
5. Фільтрування відпрацювавших комбінацій переданих. Конвертація до формату base64. Переведення до числового формату. Конвертація до великих чисел. Подання на вхід згенерованої моделі для отримання вихідних даних.
6. Тестування насіння фазингу та при повному відпрацюванні, фільтрація даних та додавання відфільтрованих даних до вхідних даних моделі.

Описані вище особливості системи генерації насіння для мутаційного фазингу дозволять, якісним чином покращити виявлення помилок при тестуванні програмного забезпечення та систематизувати підхід до генерації насіння фазингу, з підвищенням ефективності генерації, не покладаючись на рандомізовані потенційні помилки.

Висновок до розділу 3

В даному розділі був описаний метод оптимізації файлового, протокольного та мережевого мутаційного фазингу.

В результаті був представлений метод, що реалізує генерацію насіння фазингу в не залежності від того, для якого фазингу предназначний кінцевий результат моделі генерації, з внесенням особливостей на етапі генерації вхідних даних для кожного з преставлених підтипів мутаційного фазингу та преведення їх до одного формату.

Зроблено висновок, що описаний в даному розділі метод вирішує проблеми, які були в існуючих методах та збільшує загальну ефективність фазингу.

Розділ 4 Результати роботи методу генерації вхідних насінних файлів для мутаційного фазингу

В даному розділі буде представлено опис та схематична реалізація алгоритмів, що були розроблені в процесі реалізації методу генерації вхідних насінних файлів для мутаційного фазингу. Також будуть таблично представлені результати відпрацювання інструментів фазингу, що відпрацьовували на згенерованих цим методом насіннях.

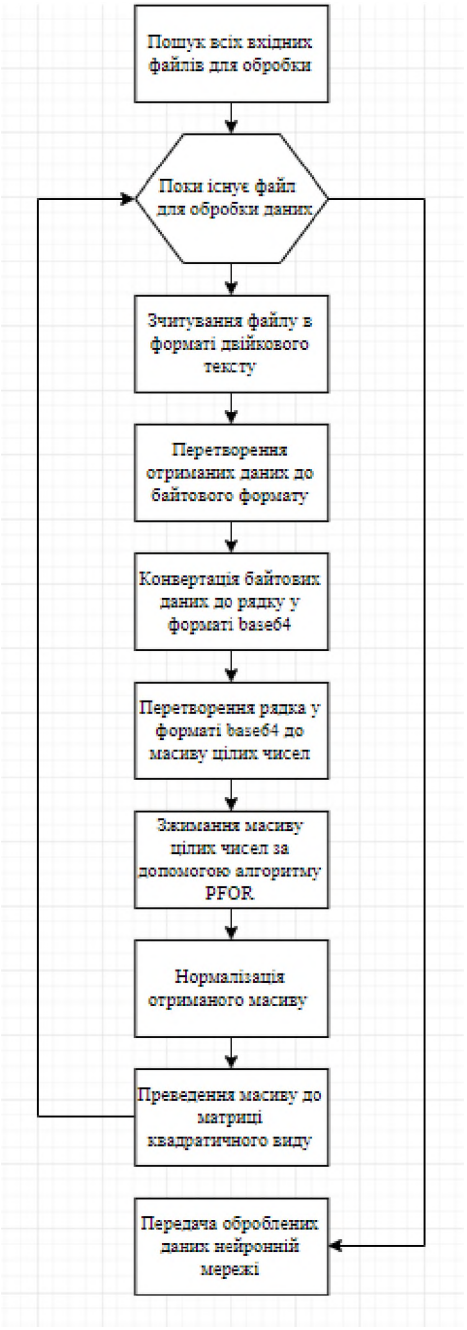

Рисунок 4.1 - Перетворення вхідних файлів інструментів фазингу на матричні дані для аналізу нейронною мережею

На основі проведеного тестування методу генерації вхідних насінних файлів для мутаційного фазингу було побудовано блок-схему(Рисунок 4.1) роботи перетворення вхідних файлів інструментів фазингу на матричні дані для аналізу нейронною мережею, лістинг коду подано у Додатку А.

Першим етапом роботи є отримання всіх вхідних файлів, що мають бути подані нейронній мережі на обробку після перетворення до відповідного формату. Наступним етапом є циклічна обробка кожного отриманого програмним забезпеченням вхідного файлу. Далі проходить етап отримання файлу програмою у двійковому форматі для подальшої обробки. Після цього отримані програмою дані перетворюються в масив байтів. Наступним етапом процесу обробки є перекодування байтового масиву у рядок формату base64, що дозволяє оброблювати всі види поданих даних в незалежності від їх форматування та складу, також на цьому етапі відбувається зменшення розміру інформації для обробки через особливість алгоритму до стиснення даних без їх втрати та з можливістю відновлення. Після цього етапу відбувається стиснення отриманого масиву цілочисельних даних за допомгою алгоритму PFOR. Наступними двома етапами є нормалізація даних та преведення їх до матриці квадратичного формату. В результаті відпрацювання програмного коду на виході отримується набір даних в дробно-чисельній формі, що подаються в нейронну мережу у вигляді матриці початкових станів.

На основі проведеного тестування методу генерації вхідних насінних файлів для мутаційного фазингу було побудовано блок-схему(Рисунок 4.2) роботи оберненого перетворення вхідних файлів інструментів фазингу на матричні дані для аналізу нейронною мережею, лістинг коду подано у Додатку Б.

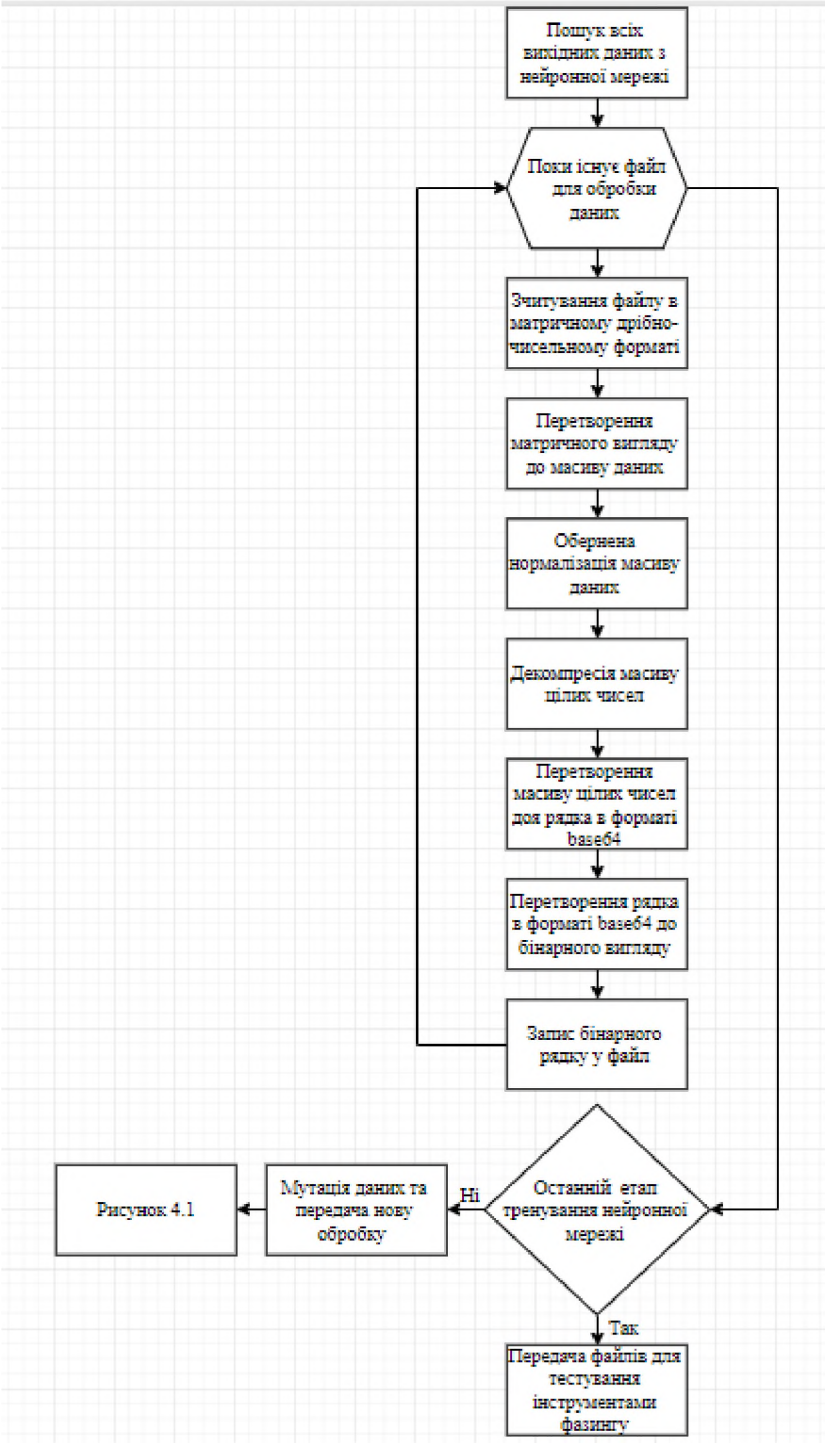

Рисунок 4.2 - Обернене перетворення вхідних файлів інструментів фазингу на матричні дані для аналізу нейронною мережею

Першим етапом роботи є отримання всіх вихідних файлів, що мають бути видані нейронною мережею після аналізу та вдосконалення для подальшої мутації та обробки нейронною мережею знову або для тестування за допомгою інструментів фазингу. Наступним етапом є циклічна обробка

кожного отриманого програмним забезпеченням вхідного файлу. Далі проходить етап отримання файлу програмою у матричному дрібно-чисельному форматі для подальшого перетворення. Після цього отримані програмою дані перетворюються в масив дрібно-чисельних даних та відбувається процес нормалізації даних у отриманому масиві. Наступним етапом процесу обробки є декомпресія масиву цілочисельних даних за допомогою оберненого алгоритму PFOR. Далі збільшений за розмірами масив подається на перетворення в рядок формату base64 з відновленням цілосності base64 коду за допомогою доповнення коду знаками “=”. Наступними двома етапами є перетворення рядка в форматі base64 до бінарного та його подальший запис у файл. В результаті відпрацювання програмного коду на виході отримується набір вхідних файлів, що в залежності від етапу тренування нейронної мережі або будуть подані на подальшу мутацію та обробку за схемою поданою у Рисунку 4.1, або будуть подані на тестування інструментами фазингу з подальшою можливістю вибору цінних вихідних даних за результатами фазингу.

На основі проведеного тестування методу генерації вхідних насінних файлів для мутаційного фазингу було побудовано блок-схему(Рисунок 4.3) роботи розбиття файлу побудови протоколу на параметри для аналізу нейронною мережею, лістинг коду подано у Додатку В.

Рисунок 4.3 - Розбиття файлу побудови протоколу на параметри для аналізу нейронною мережею

Першим етапом роботи є отримання всіх вхідних файлів, що мають бути подані нейронній мережі на обробку після перетворення до відповідного формату. Наступним етапом є циклічна обробка кожного

отриманого програмним забезпеченням вхідного файлу. Далі проходить етап отримання файлу програмою у текстовому форматі для подальшої обробки. Після цього відбувається процес пошуку входжень параметрів протоколу, що у подальшому будуть використовуватися, як елемент мутації та вдосконалення та фазингу. Наступним етапом процесу обробки є виділення серед знайдених параметрів протоколу, параметрів що будуть статичними та будуть тестуватися на наявність перевірки цілосності та дотримання формату протоколу за нормативами. Наступним етапом процесу обробки є виділення серед знайдених параметрів протоколу, параметрів що будуть динамічним та будуть основними параметрами для досліджень на можливі конфлікти в роботі протоколу при їх мутації та цілеспрямованій зміні. Наступними етапом є видалення всієї зайвої інформації, що не буде використовуватися при навчанні нейронної мережі. Далі дані, що залишилися розподіляють в два файли, що будуть окремо оброблені та протестовані. В результаті відпрацювання програмного коду на виході отримується пара наборів файлів, що містять дані які у подальшому будуть преведені до матричного дрібно-чисельного вигляду за допомогою схеми показаної на Рисунку 4.1.

На основі проведеного тестування методу генерації вхідних насінних файлів для мутаційного фазингу було побудовано блок-схему(Рисунок 4.4 – 4.5) загальної роботи методу генерації насіння мутаційного фазингу для таких підвидів фазингу, як файловий, мережевий та протокольний.

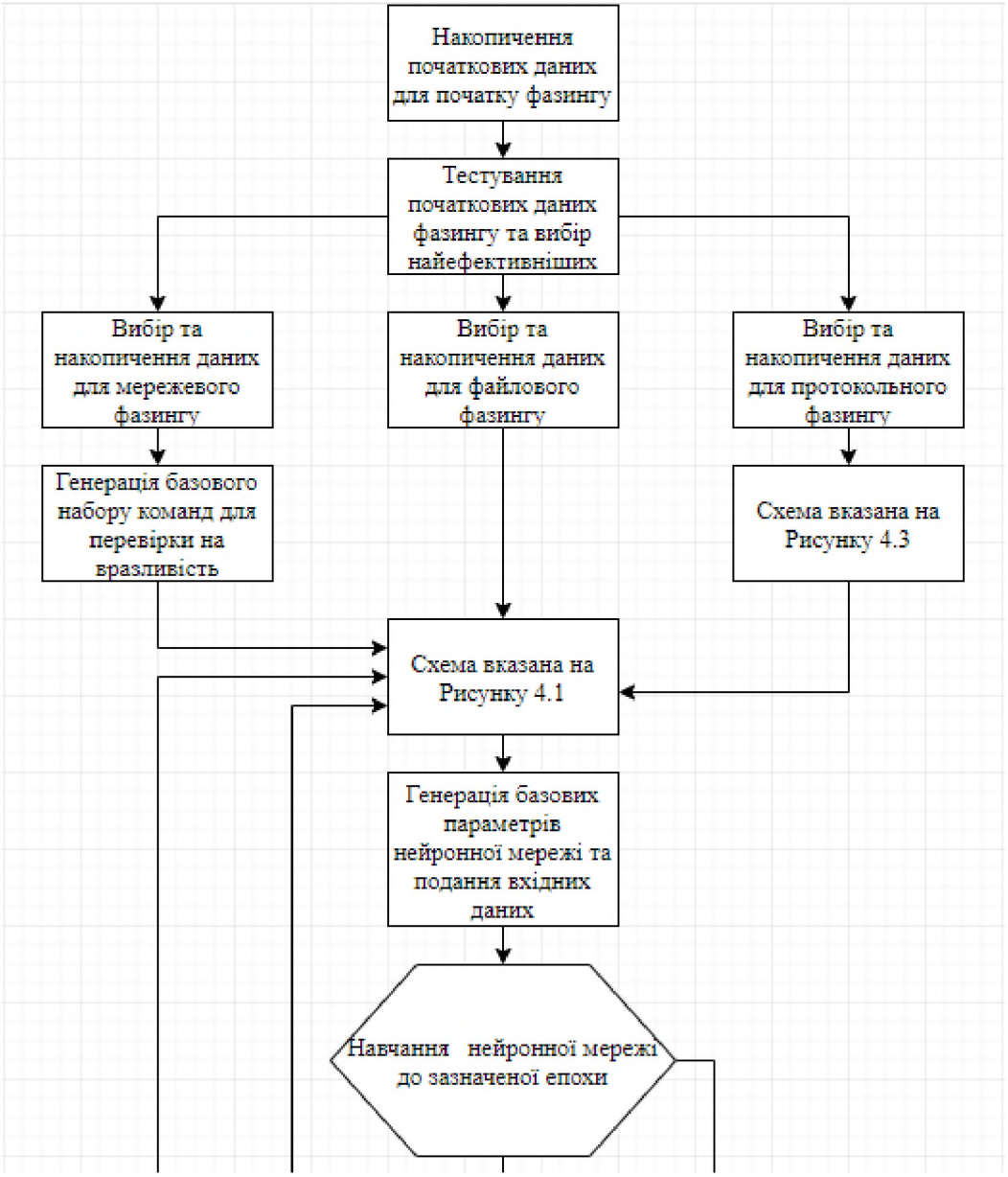

Рисунок 4.4 – Загальна схема роботи методу генерації насіння мутаційного фазингу

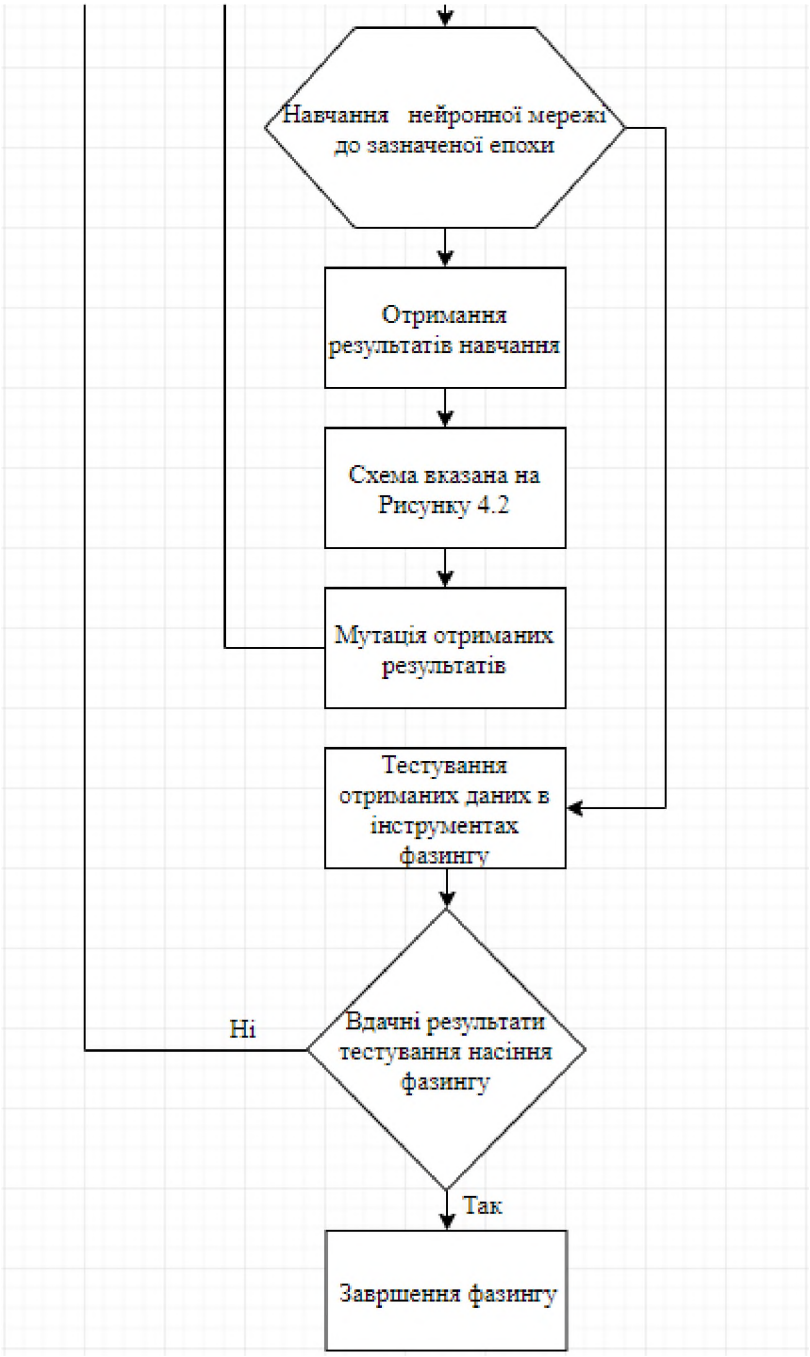

Рисунок 4.5 – Продовження загальної схеми роботи методу генерації насіння мутаційного фазингу

Першим етапом роботи є накопичення початкових файлів, що мають бути протестовані на якість потенційною генерації насіння фазингу. Наступним етапом є первірка накопичених даних та вибір набільш якісних екземплярів для подальшого тестування(Пошук балансу між розміром даних та ефективністю виявлених вразливостей). Далі проходить етап визначення для якого типу фазингу накопичувалися дані та визначення їх подальшої

обробки в залежності від отриманої інформації. Після цього відбувається процес першочергової обробки інформації для локалізації необхідних даних, як параметрів для мутації та вдосконалення. Наступним етапом процесу обробки є преведення отриманих вхідних даних до матриць з дрібно-чисельними числами, що детально показується на Рисунку 4.1. Наступним етапом є навчання нейронної мережі на отриманих даних та даних згенерованих в процесі тренування. Наступними етапом є отримання проміжних результатів навчання нейронної мережі. Далі відбувається конвертація даних за схемою детально вказаною на Рисунку 4.2. Після обробки даних в залежності від того чи настала потрібна епоха навчання дані або знову відбувається цикл обробки та навчання, або відбувається тестування отриманих даних, за допомогою інструментів фазингу. Надалі в залежності від встановлених мінімально допустимих результатів або відбувається повторне навчання вже з модифікваними даними, або процес фазингу завершується В результаті відпрацювання отримується набір вразливостей знайдених в процесі фазингу в залежності від вибраних методів мутації даних.

4.1 Результати методу генерації вхідних насінних файлів для мутаційного фазингу для файлів

4.1.1 Параметри нейронної мережі для мутаційного фазингу для файлів

На основі проведених тестувань, були вибрані такі параметри нейронної мережі, для отримання найкращих результатів.

Таблиця 4.1 – Параметри нейронної мережі

Параметр нейронної мережі	Значення параметра	
Кількість шарів нейронної мережі	3(Вхідний, Прихований, Вихідний)	
Кількість нейронів на шар	Перший шар	4 нейронів
	Другий шар	256 нейронів
	Третій шар	4 нейрон
Активаційні функції шарів	Перший шар	Лінійна функція
	Другий шар	Сигмовидна функція
	Третій шар	Сигмовидна функція
Розмір вхідних даних у бітах	64-128-192-256-2^8*k	
Алгоритм тренування нейронної мережі	(RP) TRAINRP Resilient Backpropagation	
Ідеальний результат для тренування помилки 1-го роду	5.0E-3 – 5.0E-4	
Максимальна кількість епох тренування	60000	
Параметри комп'ютера для тренування нейронної мережі	Intel® Core i7-8700K CPU 4.5 GHz, Nvidia Geforce GTX 1080, 16 GB RAM, 64-bit OS, X64-based processor (Linux)	

4.1.2 Результати відпрацювання натренованої нейронної мережі

4.1.2.1 Метод з використанням перескресної мутації

Результати знайдених вразливостей при використанні перекресної мутації:

Таблиця 4.2 – Знайдені вразливості при використанні перекресної мутації

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
mp3gain	20	50000
magick	25	50000
vlc	15	50000
Exiv2	50	50000
GZip	9	50000
LibreOffice	33	50000
DjVuLibre	7	50000
GIMP	5	50000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.3 – Перелік вразливостей при використанні перекресної мутації

Програма для тестування	Перелік вразливостей
mp3gain	CVE-2017-14410 CVE-2017-14412 CVE-2017-14406 CVE-2017-12912 CVE-2017-14409 CVE-2018-10777
magick	CVE-2018-6799 CVE-2017-16352 CVE-2017-17500 CVE-2017-12937 CVE-2017-11637
vlc	CVE-2017-8310

Продовження таблиці 4.3

Програма для тестування	Перелік вразливостей
vlc	CVE-2017-9300 CVE-2018-11516
Exiv2	CVE-2018-11037 CVE-2018-10999 CVE-2018-12264 CVE-2018-14338
GZip	CVE-2005-1228 CVE-2010-0001
LibreOffice	CVE-2017-7870 CVE-2016-10327 CVE-2018-10583 CVE-2018-10120
DjVuLibre	CVE-2012-6535
GIMP	CVE-2017-17789

4.1.2.2 Метод з використанням bitflip мутації

Результати знайдених вразливостей при використанні bitflip мутації:

Таблиця 4.4 – Знайдені вразливості при використанні bitflip мутації

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
mp3gain	13	55000
magick	20	55000
vlc	8	55000
Exiv2	33	55000
GZip	5	55000
LibreOffice	17	55000
DjVuLibre	3	55000
GIMP	2	55000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.5 – Перелік вразливостей при використанні bitflip мутації

Програма для тестування	Перелік вразливостей
mp3gain	CVE-2017-14410 CVE-2017-12912 CVE-2018-10777
magick	CVE-2018-6799 CVE-2017-16352 CVE-2017-11637
vlc	CVE-2017-8310 CVE-2018-11516
Exiv2	CVE-2018-11037 CVE-2018-10999
GZip	CVE-2005-1228
LibreOffice	CVE-2017-7870 CVE-2018-10583 CVE-2018-10120
DjVuLibre	CVE-2012-6535
GIMP	CVE-2017-17789

4.1.2.3 Метод з використанням мутації незалежних бітів

Результати знайдених вразливостей при використанні мутації 4 випадкових бітів:

Таблиця 4.6 – Знайдені вразливості при використанні мутації незалежних бітів

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
mp3gain	2	50000
magick	1	50000
vlc	0	50000
Exiv2	1	50000
GZip	3	50000
LibreOffice	2	50000
DjVuLibre	0	50000
GIMP	0	50000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.7 – Перелік вразливостей при використанні мутації незалежних бітів

Програма для тестування	Перелік вразливостей
mp3gain	CVE-2017-14412
magick	CVE-2017-12937
vlc	Не знайдено
Exiv2	CVE-2018-14338
GZip	CVE-2010-0001
LibreOffice	CVE-2018-10120
DjVuLibre	Не знайдено
GIMP	Не знайдено

4.1.3 Інтерпретація результатів роботи метода

На основі отриманих даних відпрацювання можна зробити висновок, що за наявності обмеженої кількості ресурсів за розміром вхідних даних є не ефективним використання мутації випадкових бітів, проте за часом виконання значно випереджає, як bitflip так і перекресні мутації. Найефективніше проявив себе метод з використанням перекресної мутації.

4.2 Результати методу генерації вхідних насінних файлів для мутаційного фазингу мережевих сервісів

4.2.1 Параметри нейронної мережі для мутаційного фазингу мережевих сервісів

На основі проведених тестувань, були вибрані такі параметри нейронної мережі, для отримання найкращих результатів.

Таблиця 4.8 – Параметри нейронної мережі

Параметр нейронної мережі	Значення параметра	
Кількість шарів нейронної мережі	3(Вхідний, Прихований, Вихідний)	
Кількість нейронів на шар	Перший шар	4 нейронів
	Другий шар	256 нейронів
	Третій шар	4 нейрон
Активаційні функції шарів	Перший шар	Лінійна функція
	Другий шар	Сигмовидна функція
	Третій шар	Сигмовидна функція
Розмір вхідних даних у бітах	64-128-192-256-2^8*k	
Алгоритм тренування нейронної мережі	(RP) TRAINRP Resilient Backpropagation	
Ідеальний результат для тренування помилки 1-го роду	5.0E-3 – 5.0E-4	
Максимальна кількість епох тренування	60000	
Параметри комп'ютера для тренування нейронної мережі	Intel® Core i7-8700K CPU 4.5 GHz, Nvidia Geforce GTX 1080, 16 GB RAM, 64-bit OS, X64-based processor (Linux)	

4.2.2 Результати відпрацювання натренованої нейронної мережі

4.2.2.1 Метод з використанням перескресної мутації

Результати знайдених вразливостей при використанні перекресної мутації:

Таблиця 4.9 – Знайдені вразливості при використанні перекресної мутації

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
VMware Virtual Box	10	30000
Apache Http Server	31	30000
MySQL	13	30000
Apache Tomcat	26	30000
Openssh	8	30000
Ntop	3	30000
Openldap	15	30000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.10 – Перелік вразливостей при використанні перекресної мутації

Програма для тестування	Перелік вразливостей
VMware Virtual Box	CVE-2016-5545 CVE-2017-10428 CVE-2017-10408
Apache Http Server	CVE-2018-1301 CVE-2018-8011 CVE-2017-3167 CVE-2014-0226 CVE-2017-7679
MySQL	CVE-2014-6559 CVE-2014-6496 CVE-2014-6478

Продовження таблиці 4.10

Програма для тестування	Перелік вразливостей
Apache Tomcat	CVE-2017-5650 CVE-2017-5664 CVE-2016-6817 CVE-2018-1336
Openssh	CVE-2016-6210 CVE-2017-15906
Ntop	CVE-2015-8368
Openldap	CVE-2017-9287 CVE-2015-3276 CVE-2013-4449

4.2.2.2 Метод з використанням bitflip мутації

Результати знайдених вразливостей при використанні bitflip мутації:

Таблиця 4.11 – Знайдені вразливості при використанні bitflip мутації

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
VMware Virtual Box	9	30000
Apache Http Server	27	30000
MySQL	15	30000
Apache Tomcat	18	30000
Openssh	7	30000
Ntop	1	30000
Openldap	9	30000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.12 – Перелік вразливостей при використанні bitflip мутації

Програма для тестування	Перелік вразливостей
VMware Virtual Box	CVE-2016-5545 CVE-2017-10408
Apache Http Server	CVE-2018-1301 CVE-2018-8011 CVE-2017-7679
MySQL	CVE-2014-6559 CVE-2014-6496 CVE-2014-6478
Apache Tomcat	CVE-2016-6817 CVE-2018-1336
Openssh	CVE-2017-15906
Ntop	CVE-2015-8368
Openldap	CVE-2017-9287 CVE-2015-3276

4.2.2.3 Метод з використанням мутації незалежних бітів

Результати знайдених вразливостей при використанні мутації 4 випадкових бітів:

Таблиця 4.13 – Знайдені вразливості при використанні мутації незалежних бітів

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
VMware Virtual Box	8	30000
Apache Http Server	15	30000
MySQL	9	30000
Apache Tomcat	17	30000
Openssh	5	30000
Ntop	1	30000
Openldap	4	30000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.14 – Перелік вразливостей при використанні мутації незалежних бітів

Програма для тестування	Перелік вразливостей
VMware Virtual Box	CVE-2016-5545 CVE-2017-10428
Apache Http Server	CVE-2018-1301 CVE-2017-3167 CVE-2014-0226
MySQL	CVE-2014-6559 CVE-2014-6478
Apache Tomcat	CVE-2016-6817 CVE-2018-1336
Openssh	CVE-2017-15906
Ntop	CVE-2015-8368
Openldap	CVE-2017-9287
VMware Virtual Box	CVE-2017-10408

4.2.3 Інтерпретація результатів роботи метода

На основі отриманих даних відпрацювання можна зробити висновок, що за наявності обмеженої кількості ресурсів за розміром вхідних даних є не ефективним використання мутації випадкових бітів, проте за часом виконання значно випереджає, як bitflip так і перекресні мутації. Найефективніше проявив себе метод з використанням перекресної мутації.

4.3 Результати методу генерації вхідних насінних файлів для мутаційного фазингу мережевих протоколів

4.3.1 Параметри нейронної мережі для мутаційного фазингу для мережевих протоколів

На основі проведених тестувань, були вибрані такі параметри нейронної мережі, для отримання найкращих результатів.

Таблиця 4.15 – Параметри нейронної мережі

Параметр нейронної мережі	Значення параметра	
Кількість шарів нейронної мережі	3(Вхідний, Прихований, Вихідний)	
Кількість нейронів на шар	Перший шар	30 нейронів
	Другий шар	256 нейронів
	Третій шар	30 нейрон
Активаційні функції шарів	Перший шар	Лінійна функція
	Другий шар	Сигмовидна функція
	Третій шар	Сигмовидна функція
Розмір вхідних даних у бітах	64-128-192-256-2^8*k	
Алгоритм тренування нейронної мережі	(RP) TRAINRP Resilient Backpropagation	
Ідеальний результат для тренування помилки 1-го роду	5.0E-3 – 5.0E-4	
Максимальна кількість епох тренування	60000	
Параметри комп'ютера для тренування нейронної мережі	Intel® Core i7-8700K CPU 4.5 GHz, Nvidia Geforce GTX 1080, 16 GB RAM, 64-bit OS, X64-based processor (Linux)	

4.3.2 Результати відпрацювання натренованої нейронної мережі

4.3.2.1 Метод з використанням перескресної мутації

Результати знайдених вразливостей при використанні перекресної мутації:

Таблиця 4.16 – Знайдені вразливості при використанні перекресної мутації

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
IP	3	10000
TCP	5	10000
UDP	4	10000
SSH	10	10000
DHCP	8	10000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.17 – Перелік вразливостей при використанні перекресної мутації

Програма для тестування	Перелік вразливостей
IP	CVE-2008-1095
TCP	CVE-2005-0067 CVE-2005-3675
UDP	CVE-2016-10229
SSH	CVE-2011-0766 CVE-2001-1470 CVE-2001-0259
DHCP	CVE-2012-3570 CVE-2015-8605 CVE-2009-1893

4.3.2.2 Метод з використанням bitflip мутації

Результати знайдених вразливостей при використанні bitflip мутації:

Таблиця 4.18 – Знайдені вразливості при використанні bitflip мутації

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
IP	2	10000
TCP	5	10000
UDP	2	10000
SSH	7	10000
DHCP	8	10000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.19 – Перелік вразливостей при використанні bitflip мутації

Програма для тестування	Перелік вразливостей
IP	CVE-2008-1095
TCP	CVE-2005-0067 CVE-2005-3675
UDP	CVE-2016-10229
SSH	CVE-2011-0766 CVE-2001-1470
DHCP	CVE-2012-3570 CVE-2015-8605 CVE-2009-1893

4.3.2.3 Метод з використанням мутації незалежних бітів

Результати знайдених вразливостей при використанні мутації 4 випадкових бітів:

Таблиця 4.20 – Знайдені вразливості при використанні мутації незалежних бітів

Програма для тестування	Знайдено вразливостей з унікальним кодом завершення	Всього проведено тестів
IP	0	10000
TCP	2	10000
UDP	0	10000
SSH	4	10000
DHCP	3	10000

Серед знайдених вразливостей критично небезпечними було виділено

Таблиця 4.21 – Перелік вразливостей при використанні мутації незалежних бітів

Програма для тестування	Перелік вразливостей
IP	Не знайдено
TCP	CVE-2005-3675
UDP	Не знайдено
SSH	CVE-2011-0766
DHCP	CVE-2015-8605

4.3.3 Інтерпретація результатів роботи метода

На основі отриманих даних відпрацювання можна зробити висновок, що за наявності обмеженої кількості ресурсів за розміром вхідних даних є не ефективним використання мутації випадкових бітів, проте за часом виконання значно випереджає, як bitflip так і перекресні мутації. Найефективніше проявив себе метод з використанням перекресної мутації. Загалом при тестуванні протоколів виявлено, що вразливості в сучасних версіях протоколів є доволі малоймовірними і більшість вразливостей вже були нейтралізовані протягом років використання.

Висновок до розділу 4

В даному розділі було представлено результати роботи методу оптимізації мутаційного фазингу за генерації насіння з різними методами мутації.

Зроблено висновок, що найефективнішим виявився метод з перекресною мутацією, найшвидшим за часом виконання та найгіршим за ефективністю виявився метод з мутацією незалежних рандомних бітів. Як результат були виявлені вразливості в різних версіях продуктів представлених в результатах. Слід зазначити, що сучасних версіях протоколів не було знайдено жодної вразливості, що може вказувати, як на досконаліть протоколів, що поліпшувалися десятиліттями, так й недостатню потужність обчислювальної машини для будови якісних вихідних насінь фазингу.

РОЗДІЛ 5 СТАРТАП

5.1 Опис ідеї проекту (товару, послуги, технології)

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробка або оптимізація методу мутаційної генерації вхідного насіння фазингу	1. Захист інформації	Безпека інформаційних мереж
	2. Зменшення впливу атак. Інформаційна безпека	Захищеність інформаційних мереж
	3. Виявлення вразливостей на процесі створення продукту	Економія грошей на етапі розробки та супроводження програмного забезпечення

Конкурентами є аналогічні методи мутаційної генерації вхідного насіння фазингу. Основною відмінністю є те, що метод мутаційної генерації вхідного насіння фазингу реалізується таким чином, щоб забезпечити можливість генерації насіння фазингу у загальному вигляді незалежно від того для якого типу фазингу воно в подальшому буде використано.

Довгостроковими перспективами є:

- Збільшення кількості клієнтів, що будуть використовувати запропонований метод.
- Додавання новітніх механізмів та алгоритмів мутації насіння.

Потреби в стартовому фінансуванні:

Стартовий капітал = 4000 грн

Таблиця 5.2 – Визначення сильних, слабких та нейтральних

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів				W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент1	Конкурент2	Конку-рент3			
1.	Бюджетне фінансування	розробка за рахунок розробника	розробка за рахунок бюджетних коштів	розробка комерційна	розробка за рахунок розробника	відсутність фінансування	частковабюджетне фінансування	бюджетне фінансування
2.	Використання сучасної техніки	використовується сучасна техніка	використовується застаріла техніка	використовується техніка застарілої конфігурації	використовується сучасна техніка	сучасна комплектація технікою	часткова комплектація технікою	техніка застарілої конфігурації
3.	Належна матеріально-технічна база	розробкопроводиться за власні кошти на ПК	бюджетна установа	інформаційний центр	інформаційний центр	інформаційний центр	бюджетна установа	власні кошти на приватному ПК
4.	Підключення до Інтернету	є підключення до Інтернету	є підключення до Інтернету	є підключення до Інтернету	є підключення до Інтернету	без підключення до Інтернету	часткова підключення до Інтернету	є підключення до Інтернету
5.	Налагоджена система реклами продукту	продукт не рекламується	є реклама	продукт не рекламується	є реклама	не реклама	часткова реклама	рекламується
6.	Високий рівень розробки	запропоновані методи та алгоритми є досконалими	розробконе досконала та потребує доробок	запропоновані методи та алгоритми є досконалими	розробко не досконала та потребує доробок	розробко не досконала та потребує доробок	розробко є майже досконалою	запропоновані методи та алгоритми є досконалими
7.	Професіонали програмісти	розробка проводилася студентом	розробка проводилася групою професіоналів	розробка проводилася професіоналом програмістом	розробка проводилася професіоналом програмістом	розробка проводилася студентом	розробка проводилася професіоналом програмістом	розробка проводилася групою професіоналів
8.	Налагоджена співпраця із бізнес-структурами	ні	проводиться	ні	ні	ні	частково	проводиться

5.2 Технологічний аудит ідеї проекту

Визначення технологічної здійсненності ідеї проекту передбачає аналіз таких складових (табл. 5.3):

- за якою технологією буде виготовлено товар згідно ідеї проекту?
- чи існують такі технології, чи їх потрібно розробити/доробити?
- чи доступні такі технології авторам проекту?

Таблиця 5.3 –Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Розробка або оптимізація методу мутаційної генерації вхідного насіння фазингу	Технологія 1 (технологія надання послуги)	потрібно розробити	Доступні
2		Технологія 2 (наявність бази досліджень)	наявні	Доступні
3		Технологія 3 (база проведення досліджень (випробувань))	потрібно розробити	Доступні
4		Технологія 4 (оформлення результатів дослідження)	потрібно розробити	доступні
Обрана технологія реалізації ідеї проекту: є можливою				

5.3 Аналіз ринкових можливостей запуску стартап проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	8000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	не має
5	Специфічні вимоги до стандартизації та сертифікації	Відсутні
6	Середня норма рентабельності в галузі (або по ринку), %	25

На основі проведеного дослідження є можливість стверджувати про привабливість проекту для входження на ринок за попереднім оцінюванням.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
	Виявлення якнайбільшої кількості вразливостей в товарах	Програмісти, інформаційні центрі, центри розвідки	зниження рівня загроз – системні програмісти; виявлення вразливостнй-програмісти	Якісна робота продукту. Наявність стандартизації помилок.

Таблиця 5.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Агресивність конкурентів	вплив на систему	може порушити налагоджену систему розповсюдження
2	Нестабільність політичної ситуації в світі	балансування курсу	може порушити надійну систему постачальників
3	Висока вартість продукції	підвищення ціни	підвищить агресивність конкурентів
4	Економічні складності	відсутність фінансування	порушили фінансове забезпечення компанії

Таблиця 5.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
	Тривале існування	тривале існування на ринку	на ринку дає можливість виходу на нові ринки
	Моніторинг потреб споживачів	розуміючи потреби споживачів, розширювати діапазон продукції, що випускається.	розширення діапазону продукції, що випускається.
	Лібералізація торговельних бар'єрів	робота менеджменту	призведе до поліпшення налагодженої системи розповсюдження
	Висока вартість продукції в порівнянні з ключовими конкурентами	встановлення високої ціни	утруднить вихід на нові ринки
	Стабілізація бізнес-середовища	формування стабільного середовища	за рахунок стабілізації бізнес-середовища можна поліпшити фінансове забезпечення компанії

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополія/олігополія/ монополістична/чиста	*Локальний/національний бізнес.*Глобальні сили є не досить вагомими по відношенню до локальних сил, які визначаються наявністю сертифікації, відповідності держ нормам і стандартам, регулюванням молокопереробної галузі державою.	працює в рамках розробки систем тестування програмного забезпечення

Продовження таблиці 5.8

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
2. За рівнем конкурентної боротьби - локальний/національний/…	Локальний	Ведучи конкуренцію на локальному рівні, компанії необхідно прикласти належні зусилля для охоплення всього ринку
3. За галузевою ознакою - міжгалузева/ внутрішньогалузева	*Внутрішньогалузева.* Конкуренція на ринку ведеться в інформаційній галузі України	Необхідно зосередити зусилля на пошуку конкурентних переваг, які дозволять компанії займати стійкі конкурентні позиції
4. Конкуренція за видами товарів: - товарно-родова - товарно-видова - між бажаннями	*Товарно-родова.* Конкуренція на рівні технології задоволення потреб. Існує конкуренція з іншими моделями, алгоритмами	методи мутаційної генерації вхідного насіння фазингу
5. За характером конкурентних переваг - цінова / нецінова	*Нецінова.* При виборі алгоритмів та методів споживач звертає увагу на ефективність методів оцінювання ефективності контролів безпеки. *Цінова*. Для значної частки споживачів ціна є визначальною при виборі.	Головною конкурентною перевагою є унікальність позиціонування
6. За інтенсивністю - марочна/не марочна	*Марочна.*	Диференціація методів та моделей за мотивом задоволення потреб споживачів

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Навести перелік прямих конкурентів	Визначити бар'єри входження в ринок	Визначити фактори сили постачальників	Визначити фактори сили споживачів	Фактори загроз з боку замінників

Продовження таблиці 5.9

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Висновки:	На ринку спостерігається тенденція до скорочення кількості підприємств і посилення конкуренції на ринку. Вступ України до СОТ відкрив дорогу іноземним виробникам. Великі компанії з іноземним капіталом постійно збільшують контрольовану ними частку ринку, поглинаючи конкурентів.	Бар'єри входу на ринок є порівняно незначними. Вартість організації бізнесу з виробництва сучасних методів мутаційної генерації вхідного насіння фазингу сягає 10 тис. дол.	Існує чітка залежність від постачальників як якості продукції. Також ціна кінцевої продукції залежить від рівня сертифікації.	Споживачі мають широку географію і проживають переважно у містах. Покупка програмних додатків та алгоритмів мутаційної генерації вхідного насіння фазингу часто носить імпульсний характер.	Посилилася конкуренція зі сторони товарів субститутів – інших видів методів та алгоритмів методу мутаційної генерації вхідного насіння фазингу, за рахунок збільшення асортименту останніх та появи нових для ринку категорій.

Отже, відповідно до наведеного вище аналізу головними силами, які діють на конкуренцію в галузі є постачальники та споживачі. Також в силу розвитку ринку все більшого значення набуває інтенсивність конкуренції між існуючими конкурентами та загроза зі сторони товарів-субститутуів.

Таким чином в межах структурного підходу до аналізу конкуренції тип конкуренції – *монополістична конкуренція*.

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспромо жності	Обґрунтування вибору
1	Частка ринку	Враховуючи той факт, що тип родового середовища в галузі – консолідований ринок, тобто існує група компаній, які контролюють разом понад 40% ринку, а також те, що інтенсивність суперництва між діючими конкурентами при низьких темпах зростання ринку є однією з головних сил, які діють на конкуренцію в галузі, одним з найважливіших факторів конкурентоспроможності виступає частка ринку, яку займає виробник. В таких умовах чим більше частка ринку, тим більшими ринковими можливостями володіє виробник.
2	Ціна	Чим вигіднішою є ціна для споживача, тим вірогідніше його вибір.
3	Асортимент	В умовах збільшення інтенсивності між існуючими конкурентами завоювання споживачів відбувається за рахунок нових методів та алгоритмів.
4	Доступ до каналів розподілу	Споживач далеко не завжди проявляє прихильність до певної категорії розробників і дуже схильний до експериментів. В цьому випадку завоювати лояльність споживача дуже складно і ще складніше її утримати. Тому для компаній-виробників ключовими чинниками успіху стає сильна дистрибуція, якісний торговий маркетинг і налагоджена система логістики.
5	Торговий маркетинг	
6	Рівень диференціації ТМ	В умовах ведення конкурентної боротьби на споживчому ринку, де попит є ірраціональним та існує велика кількість виробників і розробників при фактично відсутній різниці між товарами, що пропонуються, ключовим фактором успіху є здатність чітко диференціювати ТМ від ТМ конкурентів, надаючи споживачеві унікальну цінність.

Продовження таблиці 5.10

№	Фактор конкурентоспромо жності	Обґрунтування вибору
7	Репутація виробника	Якщо компанія має бездоганну репутацію, особливо у сфері якості своєї продукції, то рівень довіри до неї зростає. Також репутація виробника важлива при виході на ринок з новими товарами, або при виході на нові сегменти, що полегшує позитивне сприйняття новинок.
8	Рівень лояльності до бренду	Чим вище рівень лояльності, тим більше компанія має прихильних, а значить постійних споживачів.
9	Унікальність позиціонування	В умовах монополістичної конкуренції, коли фактор диференціації ТМ є ключовим засобом ведення конкурентної боротьби, важливим є створення та підтримання унікального позиціонування, що створює певний захист від конкурентних зіткнень.
10	Маркетинговий бюджет	Від розміру маркетингового бюджету залежить здатність здійснювати маркетингову стратегію підприємства. Маркетингові заходи мають забезпечувати інші конкурентні переваги такі, як рівень диференціації, лояльності, репутація виробника, дистрибуція та просування в торгових точках.

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін «Метод оцінювання ефективності контролів безпеки»

№	**Фактор конкурентоспроможності**	**Вагові значення фактора (1-20)**	**Рейтинг конкурентів у порівнянні з метод оцінювання ефективності контролів безпеки**						
			-3	**-2**	**-1**	**0**	**1**	**2**	**3**
1	Частка ринку	20		II	III			I	
2	Ціна	10		II		III	I	I	
3	Асортимент	18		III		II	I		
4	Доступ до каналів розподілу	15	I		II				III
5	Торговий маркетинг	15		I	III				II
6	Рівень диференціації ТМ	13	III		I			II	
7	Репутація виробника	12		I		III		II	
8	Рівень лояльності до бренду	14		I			II	III	
9	Унікальність позиціонування	15				II		III	I
10	Маркетинговий бюджет	10	II			III	I		

Умовні позначки позицій конкурентів:

І - конкурент 1

ІІ - конкурент 2;

ІІІ - конкурент 3.

Отже, відповідно до проведеного аналізу можна сказати, що «Метод мутаційної генерації вхідного насіння фазингу» має наступну позицію на ринку:

сильні сторони:

– унікальне позиціонування;
– значний рівень диференціації ТМ;
– позитивна репутація виробника;

слабкі сторони:

– вища ціна порівняно з конкурентами;
– торговий маркетинг.

Виділивши найвагоміші сильні та слабкі сторони « Метод мутаційної генерації вхідного насіння фазингу» у порівнянні з основними конкурентами і з аналізу внутрішніх факторів та використовуючи результати аналізу маркетингових загроз та можливостей, складемо матрицю SWOT-аналізу (табл. 5.12.).

Таблиця 5.12 – SWOT-аналіз стартап-проекту

Сильні сторони	Слабкі сторони
1. унікальне позиціонування; **2.** значний рівень диференціації **3.** позитивна репутація виробника; **4.** приналежність до української міжнародної компанії; **5.** налагоджена система дистрибуції товару; **6.** наявність вертикальної інтеграції.	**1.** вища ціна порівняно з конкурентами. **2.** залежність маркетингової політики від українського власника; **3.** слабке самозабезпечення фінансовими ресурсами; **4.** відсутність чітко вираженої маркетингової стратегії, непослідовність в її реалізації.
Можливості	**Загрози**
1. Можливість зміцнення іміджу **2.** Можливість збільшення обсягів реалізації **3.** Можливість збільшення обсягів продаж за рахунок експансії в регіони	**1.** Загроза працювати без прибутку скорочення платоспроможного попиту **2.** Загроза втрати споживачів внаслідок підвищення тиску зі сторони товарів-субститутів **3.** Загроза підвищення цін

З результатів SWOT-аналізу видно, що найбільш негативний вплив на діяльність «Метод мутаційної генерації вхідного насіння фазингу» на ринку чинить ринкове середовище. Це, перш за все, пов'язано із наслідками фінансово-економічної кризи в країні.

В свою чергу, така ситуація супроводжувалася зменшенням темпів приросту галузі, виходом з ринку менш сильних дрібних та регіональних виробників, приходом на ринок транснаціональних компаній, що збільшило інтенсивність конкуренції між діючими учасниками ринку України.

Було визначено, що найбільшою загрозою для «Метод мутаційної генерації вхідного насіння фазингу» є загроза падіння прибутковості внаслідок скорочення попиту.

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Використання засобів стимулювання збуту та мерчандайзингу в торгових точках для збільшення продаж	Дозволяє суттєво збільшити обсяги продаж	до місяця
2.	Розширення асортиментної лінійки	Можливість залучення нових споживачів за рахунок новинки	до пів року
3.	Збільшення представленості	Можливість розширення охоплення цільової аудиторії	до року

5.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: опис цільових груп потенційних споживачів (табл.5.14).

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Метод мутаційної генерації вхідного насіння фазингу	готовий	Високий	Середня	складний
2	Зменшення впливу атак у безпроводових мережах	готовий	Високий	максимальна	простий
3	Підвищення ефективності мутаційної генерації вхідного насіння фазингу	готовий	Високий	Середня	складний

За результатами аналізу потенційних груп споживачів (сегментів) обрано стратегію диференційованого маркетингу.

Таблиця 5.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку*
1	**Стратегія диференціації**	передбачає надання товару важливих з точки зору споживача відмітних властивостей, які роблять товар відмінним від товарів конкурентів. Така відмінність може базуватися на об'єктивних або суб'єктивних, відчутних і невідчутних властивостях товару(у ширшому розумінні – комплексі маркетингу), бути реальною або уявною.	Реалізація цієї стратегії вимагає, як правило, більш високих витрат. Проте успішна диференціація дозволяє компанії домогтись більшої рентабельності за рахунок того, що ринок готовий прийняти більш високу ціну (цінову премію бренду).	Інструментом реалізації стратегії диференціації є ринкове позиціонування.

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
	Ні	к залучати нових так і забирати існуючих у конкурентів	частково	наслідування лідеру

Таблиця 5.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Відповідність чинним нормативам	Наслідування лідеру	Реалізація цієї стратегії вимагає, як правило, більш високих витрат. Проте успішна диференціація дозволяє компанії домогтись більшої рентабельності за рахунок того, що ринок готовий прийняти більш високу ціну (цінову премію бренду).	Унікальність Доступна ціна Реалізація нових методів

5.5 Розроблення маркетингової програми

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. 4.18 підсумовуємо результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.28 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1.	Виявлення вразливостей в ПО	Усунення вразливостей	високий показник ефективності виявлення загроз
2.	Виявлення вразливостей в мережах	Захищеність мереж	високий рівень виявлення вразливих місць протоколів та мережевих служб
3.	Загальний метод фазингу	Можливість використання насіння фазингу в будь-якому продукті фазингу	Загальна структура насіння для всіх форматів вхідниз даних

Таблиця 5.19 – Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Опис базової потреби споживача, яку задовольняє товар (згідно концепції), її основної функціональної вигоди		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	зниження рівня загроз – системні програмісти; виявлення вразливостнй-програмісти		
	Пакування – без пакування		
	Марка: Метод мутаційної генерації вхідного насіння фазингу		
III. Товар із підкріпленням	До продажу – рівень розробки		
	Після продажу – низка методів та алгоритмів		
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності			

Таблиця 5.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
	8000	7500	11000	2000-3000

Таблиця 5.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
	Мінімальна кількість посередників	організовувати широку мережу збуту товару	3	непряма

Таблиця 5.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Метод мутаційної генерації вхідного насіння фазингу	інформаційні мережі	Алгоритми генерації вхідного насіння фазингу	донести переваги до потенційних користувачів	Основна ідея Генерація вхідного насіння фазингу
2	Захищеність мереж	інформаційні мережі	Виявлення вразливостей в мережах та протоколах	донести переваги до потенційних користувачів	
3	Якість виявлення вразливостей в ПО	інформаційні мережі	підвищення ефективності генерації вхідного насіння фазингу	донести переваги до потенційних користувачів	

Висновок до розділу 5

В умовах розділу проведено аналіз та розробку бізнес-проекту до розробки «Метод мутаційної генерації вхідного насіння фазингу», на основі проведеного аналізу варто відзначити, що найбільш негативний вплив на діяльність «Метод мутаційної генерації вхідного насіння фазингу» на ринку чинить ринкове середовище. Це, перш за все, пов'язано із наслідками фінансово-економічної кризи в країні. В свою чергу, така ситуація супроводжувалася зменшенням темпів приросту галузі, виходом з ринку менш сильних дрібних та регіональних виробників, приходом на ринок транснаціональних компаній, що збільшило інтенсивність конкуренції між діючими учасниками ринку України. Було визначено, що найбільшою загрозою для «Метод мутаційної генерації вхідного насіння фазингу» є загроза падіння прибутковості внаслідок скорочення попиту.

ВИСНОВКИ

Результатом даної роботи є метод мутаційного фазингу. Перевагою перед існуючими методами є побудова загальної структури вхідних та вихідних данних в незалежності від виду фазингу та оптимізація генерації насіння на основі запропонованої моделі представлення даних фазингу.

Була проведена практична реалізація методу для трьох видів фазингу:

- Файлового
- Протокольного
- Мережевого

Результати роботи:

Найефективнішим виявився метод з перекресною мутацією, найшвидшим за часом виконання та найгіршим за ефективністю виявився метод з мутацією незалежних рандомних бітів. Як результат були виявлені вразливості в різних версіях продуктів представлених в результатах. Слід зазначити, що сучасних версіях протоколів не було знайдено жодної вразливості, що може вказувати, як на досконаліть протоколів, що поліпшувалися десятиліттями, так й недостатню потужність обчислювальної машини для будови якісних вихідних насінь фазингу.

З поданих результатів видно, що даний метод не є ідеальними, можливі проблеми при виявленні вразливостей в різних версіях програмного та мережевого забезпечення, проте достовірно відомо, що даний виявляє вже знайдені вразливості попередніх версій програмного та мережевого забезпечення, а отже його ефективність як метода тестування на вразливості є високою. Вирішує проблему генерації насіння фазингу, при якій для кожного виду фазингу потрібно було використовувати окремий метод генерації насіння в залежності від різноманітності вхідних даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1 Fuzzing base information [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Fuzzing

2 Fuzzing types identification [Електронний ресурс] – Режим доступу до ресурсу: https://www.owasp.org/index.php/Fuzzing

3 Wilson T. Evaluation of Fuzzing as a Test Method for an Embedded System[C].Vaasa, 2018:1-3

4 Godefroid P, Kiezun A, Levin M. Grammar-based Whitebox Fuzzing[C]. ASE, 2016:1-2.

5 Godefroid P, Molnar D, Levin M. SAGE: Whitebox Fuzzing for Security Testing[C]. ACMQUEUE, 2012:3-5.

6 AnsVif fuzzer tool [Електронний ресурс] – Режим доступу до ресурсу: https://oxagast.github.io/ansvif/

7 Hjelt I. The future of grey-box fuzzing[C]. VT,2017:9-13

8 Wustholz V, Christakis M. Learning Inputs in Greybox Fuzzing[C]. ArXiv, 2018:1-2

9 Sanjay R, Vivek J, Kumar A, Cojocar L, Giuffrida C, Bos H. VUzzer: Application-aware Evolutionary Fuzzing[C]. NDSS, 2017: 4-7

10 Godefroid P, Peleg H, Singh R. Learn&Fuzz: Machine learning for input fuzzing[C]. ASE, 2017:50-59.

11 Peach fuzzer tool [Електронний ресурс] – Режим доступу до ресурсу: http://www.trinity.co.il/peach-fuzzer/

12 Royal M, Pokorny P. Dumb Fuzzing in Practice[C]. Capstone, 2012:2-4

13 AFL fuzzer tool [Електронний ресурс] – Режим доступу до ресурсу: http://lcamtuf.coredump.cx/afl/README.txt

14 Shastry B, Maggi F, Yamaguchi F, Rieck K, Seifert J-P. Static Exploration of Taint-Style Vulnerabilities Found by Fuzzing[C]. FTR, 2017: 4-5

15 Wang J, Chen B, Wei L, Liu Y. Skyfire: Data-Driven Seed Generation for Fuzzing[C]. SP, 2017:579–594

16 Agrawal A, Khan R.A. A Framework to Detect and Analyze Software Vulnerabilities[C]. DIT, 2009:1-2

17 Definitions of mutational fuzzing [Електронний ресурс] – Режим доступу до ресурсу: https://www.mwrinfosecurity.com/our-thinking/15-minute-guide-to-fuzzing/

18 Autofuzz fuzzer tool [Електронний ресурс] – Режим доступу до ресурсу: http://autofuzz.sourceforge.net/developersGuide.html

19 Miller C, Peterson Z. Analysis of mutation and Generation-Bassed Fuzzing[C], Independent Security Evaluators, 2007: 3-6

20 Peach fuzzer tool [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/MozillaSecurity/peach/blob/master/README.md

21 Peach fuzzer basic tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.mozilla.org/Security/Fuzzing/Peach

22 Chenyang L, Shouling J, Yuwei L, Junfeng Z, Jianhai C, Pan Z, Jing C. Smart Seed Generation for Efficient Fuzzing[C]. CoRR, 2018: 2-4

23 Woo M, Kil C S, Gottlieb S, Brumley D. Sheduling of Mutational Fuzing Seed. ACM SIGSAC, 2013:2-5

24 Sang K C, Woo M, Brumley D. Program-Adaptive Mutational Fuzzing[C] S&P, 2015:725-741.

25 Rebert A, Sang K C, Grieco G, et al. Optimizing seed selection for fuzzing[C]. Usenix, 2014:861-87

Додаток А

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using CSharpFastPFOR.Differential;
using CSharpFastPFOR.Port;
using CSharpFastPFOR.Tests.Port;
using Xunit;
namespace FileToFlowBigInteger
{
   class Program
   {
      static string ccArr;
      static void Main(string[] args)
      {
         GenerateBasicConversionRatio();
         foreach(string Data in Directory.GetFiles("Data"))
         {
            string bsTemp = ConVertToBinaryString(Data);
```

```
            byte[] bbTemp = GetBytes(bsTemp);
            string b64sTemp = BinaryStringToBase64(bbTemp);
            int[] iarrTemp = Base64StringToIntegerArray(b64sTemp);
            int[][] imatTemp = IntegerArrayToDecimalMatrix(iarrTemp);
            using (StreamWriter outputFile = new
StreamWriter(Path.Combine(Directory.GetCurrentDirectory(),"MatrixData",
Path.GetFileName(Data))))
            {
               for (long i = 0; i < imatTemp.Length; ++i)
                  for (long j = 0; j < imatTemp.Length; ++j)

outputFile.WriteLine(imatTemp[i][j]/Math.Pow(10,Convert.ToString(imatTemp[i]
[j]).Length));
            }
         }
      }
      private static readonly IntCompressor[] ic = {
         new IntCompressor(new VariableByte()),
         new IntCompressor(new SkippableComposition(new BinaryPacking(), new
VariableByte())) };
      static string ConVertToBinaryString(string Data)
      {
         FileStream fTemp = File.Open(Data,FileMode.Open,FileAccess.Read);
         BinaryReader bin = new BinaryReader(fTemp);
```

```
        byte[] bTemp = bin.ReadBytes(Convert.ToInt32(fTemp.Length));
        return string.Join(" ", bTemp.Select(bit => Convert.ToString(bit,
2).PadLeft(8, '0')));
    }
    static string BinaryStringToBase64(byte[] Data)
    {
        long arrLength = (long)(4.0d * Data.Length / 3.0d);
        if ((arrLength % 4) != 0)
        {
            arrLength += 4 - (arrLength % 4);
        }
        char[] encodedCharArray = new char[arrLength];
        Convert.ToBase64CharArray(Data, 0, Data.Length, encodedCharArray, 0);

        return (new string(encodedCharArray));
    }
    static byte[] GetBytes(string str)
    {
        byte[] bytes = new byte[str.Length * sizeof(char)];
        System.Buffer.BlockCopy(str.ToCharArray(), 0, bytes, 0, bytes.Length);
        return bytes;
    }
    static void GenerateBasicConversionRatio()
```

```
{
    for (int i = 0; i < 26; ++i)
        ccArr += (char)(i + 65);
    for (int i = 26; i < 52; ++i)
        ccArr += (char)(i + 71);
    for (int i = 52; i < 62; ++i)
        ccArr += (char)(i - 4);
    ccArr +='+';
    ccArr +='/';
    ccArr +='=';
}
static int[] Base64StringToIntegerArray(string b64string)
{
    int[] iarrTemp = new int[b64string.Length];
    int count=0;
    foreach(char b64 in b64string)
    {
        iarrTemp[count++] = ccArr.IndexOf(b64);
    }
    return iarrTemp;
}
static int[][] IntegerArrayToDecimalMatrix(int[] imatTemp)
```

```
{
    int imatTempSize = imatTemp.Length / 6 + 1;
    int imatSize = (int)Math.Sqrt(imatTempSize) + 1;
    long[][] iMat = new long[imatSize][];
    for (long i = 0; i < imatSize;++i )
        iMat[i] = new long[imatSize];
    for (long i = 0; i < imatSize; ++i)
        for (long j = 0; j < imatSize; ++j)
            iMat[i][j] = 0;
    int count=0,row=0,count2=0;
    while(count+row*imatSize<imatTempSize)
    {
        foreach (IntegratedIntCompressor j in ic)
            iMat[row][count] = j.compress(imatTemp, count2, 6);
        count++;
        count2+=6;
        if(count==imatSize)
        {
            count=0;
            row+=1;
        }
    }
```

```
                    foreach (IntegratedIntCompressor j in ic)
                        iMat[row][count] = j.compress(imatTemp, count2,
imatTemp.Length % 6);
            return iMat;
        }
    }
}
```

Додаток Б

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using CSharpFastPFOR.Differential;
using CSharpFastPFOR.Port;
using CSharpFastPFOR.Tests.Port;
using Xunit;
namespace FileToFlowBigInteger
{
   class Program
   {
      static string ccArr;
      static void Main(string[] args)
      {
         GenerateBasicConversionRatio();
         foreach (string Data in Directory.GetFiles("MatrixData"))
         {
            double[] darrTemp;
```

```
            using (StreamReader inputFile = new
StreamReader(Path.Combine(Directory.GetCurrentDirectory(),
"MatrixDataIntegrated", Path.GetFileName(Data))))
            {
                int
arrLen=inputFile.ReadToEnd().Length*inputFile.ReadToEnd().Length;
                darrTemp = new double[arrLen];
                string line;
                int cou=0;
                while ((line = inputFile.ReadLine()) != null)
                {
                    string[] doubles = line.Split(' ');
                    foreach (string Double in doubles)
                    {
                        double value;
                        if (!double.TryParse(Double, out value))
                        {
                            Console.WriteLine("Bad value");
                        }
                        else
                            darrTemp[cou++]=value;
                    }
                }
```

```
            }
            int[] iarrDecode = Decode(darrTemp);
            string b64sTemp = Base64StringToDoubleArray(iarrDecode);
            byte[] bbTemp = ConVertToBinaryString(b64sTemp);
            using (BinaryWriter writer = new
BinaryWriter(File.Open(Path.Combine(Directory.GetCurrentDirectory(),
"MatrixDataIntegrated", Path.GetFileName(Data)), FileMode.Create)))
            {
                writer.Write(bbTemp);
            }
        }
    }
    private static readonly IntCompressor[] ic = {
        new IntCompressor(new VariableByte()),
        new IntCompressor(new SkippableComposition(new BinaryPacking(), new
VariableByte())) };
    static byte[] ConVertToBinaryString(string Data)
    {
        for (int i = 0; i < ((4 - (Data.Length % 4)) % 4); ++i)
            Data += "=";
        return Convert.FromBase64String(Data);
    }
    static void GenerateBasicConversionRatio()
```

```
{
    for (int i = 0; i < 26; ++i)
        ccArr += (char)(i + 65);
    for (int i = 26; i < 52; ++i)
        ccArr += (char)(i + 71);
    for (int i = 52; i < 62; ++i)
        ccArr += (char)(i - 4);
    ccArr += '+';
    ccArr += '/';
    ccArr += '=';
}
static string Base64StringToDoubleArray(int[] dArr)
{
    string sConcat=null;
    foreach (int sym in dArr)
    {
        sConcat += ccArr[sym];
    }
    return sConcat;
}
static int[] Decode(double[] dArr)
{
```

```
            int[] dTemp = new int[dArr.Length*6];
            int[] iArr = new int[dArr.Length];
            for (long i = 0; i < dArr.Length; ++i)
               iArr[i] = (int)(dArr[i] * Math.Pow(10,
Convert.ToString(dArr[i]).Length));
            int count = 0;
            foreach (IntegratedIntCompressor j in ic)
            {
               int[] temp = j.uncompress(iArr, count);
               dTemp[count * 6] = temp[0];
               dTemp[count * 6+1] = temp[1];
               dTemp[count * 6+2] = temp[2];
               dTemp[count * 6+3] = temp[3];
               dTemp[count * 6+4] = temp[4];
               dTemp[count * 6+5] = temp[5];
            }
            return dTemp;
         }
      }
   }
```

Додаток В

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
namespace ParseProtocol
{
   static class Program
   {
      static void Main(string[] args)
      {
         foreach (string Data in Directory.GetFiles("ProtocolData"))
         {
            FileStream fTemp = File.Open(Data, FileMode.Open, FileAccess.Read);
            StreamReader sr = new StreamReader(fTemp);
            string data = sr.ReadToEnd();
            sr.Close();
            IEnumerable<int> result = AllIndexesOf(data, " = ");
            string[] StaticParameter=new string[result.Count()],
DynamicParameter=new string[result.Count()];
```

```
            int count=0;
            foreach (int res in result)
            {
                int leftCount=res-1, rightCount=res+3;
                while (data[leftCount] != ' ' && data[leftCount] != '\r' &&
data[leftCount] != '\n')
                    leftCount--;
                while (data[rightCount] != ' ' && data[rightCount] != '\r' &&
data[rightCount] != '\n')
                    rightCount++;
                StaticParameter[count] = data.Substring(leftCount + 1, res -
leftCount);
                DynamicParameter[count] = data.Substring(res+3, rightCount-res-3);
                count++;
            }
            using (StreamWriter outputFile = new
StreamWriter(Path.Combine(Directory.GetCurrentDirectory(),
"ParsedProtocolData", Path.GetFileNameWithoutExtension(Data) +
"Static"+Path.GetExtension(Data)),false))
            {
                foreach (string par in StaticParameter)
                    outputFile.WriteLine(par);
            }
```

```
            using (StreamWriter outputFile = new
StreamWriter(Path.Combine(Directory.GetCurrentDirectory(),
"ParsedProtocolData", Path.GetFileNameWithoutExtension(Data) +
"Dynamic"+Path.GetExtension(Data)),false))
            {
                foreach (string par in DynamicParameter)
                    outputFile.WriteLine(par);
            }
        }

    }
    public static IEnumerable<int> AllIndexesOf(this string str, string
searchstring)
    {
        int minIndex = str.IndexOf(searchstring);
        while (minIndex != -1)
        {
            yield return minIndex;
            minIndex = str.IndexOf(searchstring, minIndex + searchstring.Length);
        }
    }
  }
}
```